VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Barbora Kabelková

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

GENEROVÁNÍ PRVOČÍSEL POMOCÍ HARDWARE
HARDWARE GENERATION OF CRYPTOGRAPHIC-SAFE PRIMES.

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Barbora Kabelková

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Peter Cíbik

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Studentka: Barbora Kabelková ***ID:*** 209704

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Generování prvočísel pomocí hardware

POKYNY PRO VYPRACOVÁNÍ:

Téma práce je zaměřené na návrh a implementaci generátoru kryptograficky bezpečných prvočísel na platformě FPGA. Nastudujte a teoreticky popište problematiku prvočísel a seznamte se s platformou FPGA a jazykem VHDL. Dále prostudujte možnostmi a způsoby generování prvočísel a doposud známé hardwarové implementace. Zvolte Vámi vybraný způsob a navrhnete jeho implementaci pro platformu FPGA. Tento návrh následně implementujte v jazyce VHDL. Výstupem bakalářské práce bude funkční a otestovaná hardwarová implementace generátoru kryptograficky bezpečných prvočísel na platformě FPGA v jazyce VHDL. V závěru diskutujte dosažené výsledky celého návrhu.

DOPORUČENÁ LITERATURA:

[1] PINKER, Jiří a Martin POUPA. Číslicové systémy a jazyk VHDL. Praha: BEN - technická literatura, 2006. ISBN 80-7300-198-5

[2] BURDA, Karel. Aplikovaná kryptografie. Brno: VUTIUM, 2013. ISBN 978-80- 214-4612-0

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. Peter Cíbik

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá problematikou prvočísel a jejich generováním. Stručně definuje prvočísla a poukazuje na význam bezpečných prvočísel v kryptografii. Uvádí příklady asymetrických šifer a podrobně rozebírá algoritmus RSA. Dále představuje vybrané pseudonáhodné a náhodné metody generování posloupností čísel a porovnává jejich vlastnosti. Hodnotí nejznámější pravděpodobnostní a skutečné testy prvočíselnosti na základě efektivity jejich aplikace v praxi. Navrhuje různé kombinace těchto testů s metodami generování a vybírá z nich jednu k implementaci na platformě FPGA. Práce popisuje implementaci generátoru, který využívá von Neumannovu metodu středních řádů pro vygenerování posloupnosti čísel, a následně Miller-Rabinovým testem vyhodnocuje, která z generovaných čísel jsou prvočísly. Slovně i schematicky jsou rozebrány nejdůležitější procesy, které takto navržený generátor vykonává. Návrh generátoru je simulován a syntetizován v prostředí Xilinx Viavado. Jednotlivé části generátoru jsou otestovány pomocí několika behaviorálních simulací. Na závěr práce komentuje průběh simulací a hodnotí vlastnosti navržené implementace generátoru.

KLÍČOVÁ SLOVA

asymetrická šifra, FPGA, hardwarový generátor, kryptografický algoritmus, PRNG, prvočíslo, test prvočíselnosti, TRNG, VHDL

ABSTRACT

The bachelor's thesis deals with the topic of prime numbers and their generation. It briefly introduces prime numbers and points out the importance of secure primes in cryptography. It gives examples of asymmetric ciphers and closely analyses RSA algorithm. The thesis then presents some pseudo-random and true-random methods of generating sequences of numbers and compares their properties. It evaluates the most used primality tests, both probabilistic and real, based on their applicability in practice. It suggests several combinations of primality tests with generating methods and chooses one to implement on FPGA. The thesis describes the implementation of a generator that generates a sequence of numbers using the von Neumann middle-square method and subsequently uses the Miller-Rabin test to find primes between those numbers. Key processes of the proposed generator are explained and illustrated. The proposed implementation is simulated and synthesized in the Xilinx Viavado environment. The individual parts of the generator are tested using several behavioral simulations. Finally, the thesis comments on the conducted simulations and evaluates the properties of the proposed implementation.

KEYWORDS

asymmetric cipher, FPGA, hardware generator, cryptographic algorithm, PRNG, prime number, prime test, TRNG, VHDL

KABELKOVÁ, Barbora. *Generování prvočísel pomocí hardware*. Brno, 2021, 58 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Peter Cíbik

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Generování prvočísel pomocí hardware“ jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autorky

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu bakalářské práce panu Ing.Peterovi Cíbikovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Bakalářská práce se věnuje problematice generování kryptograficky bezpečných prvočísel a návrhu hardwarového generátoru.

Zahrnuje teoretický úvod do oblasti prvočísel a demonstruje jejich důležitost v asymetrické kryptografii. Uvádí několik příkladů šifer, které se bez prvočísel neobejdou. Podkapitola o šifrovacím algoritmu RSA obsahuje i názorný příklad, na kterém je ukázáno, že příliš nízká prvočísla zásadně ohrožují bezpečnost takovýchto algoritmů.

Dále práce vysvětluje rozdíl mezi náhodnými a pseudonáhodnými generátory a rozebírá několik algoritmů pro generování pseudonáhodných posloupností. Následně se věnuje skutečným a pravděpodobnostním testům, konkrétně Fermatovu, Miller-Rabinovu a Lucas-Lehmerovu testu.

Práce navrhuje možnosti aplikace testů prvočíselnosti na náhodně vygenerované posloupnosti přirozených čísel za účelem efektivního generování prvočísel pro potřeby bezpečných asymetrických šifer. Představuje několik kombinací nástrojů, které by mohly být implementovány na platformě FPGA a na základě nastudované literatury vybírá postup slibující nejlepší vlastnosti generátoru.

V předposlední kapitole je navržen zdrojový kód pro implementaci generátoru v jazyce VHDL. Tento kód byl syntetizován v programu Xilinx Vivado a bylo provedeno několik behaviorálních simulací. Na verzi pro 16bitová čísla je otestována funkčnost navrhované implementace. Nakonec práce diskutuje o vlastnostech simulované implementace a výsledek je porovnán s očekáváním založeným na nastudované literatuře.

1 Význam prvočísel v kryptografii

Správné fungování informačních a komunikačních systémů se neobejde bez kryptografie. Většina dnes běžně používaných šifrovacích algoritmů chrání citlivá data všeho druhu, zajišťuje soukromí v komunikaci uživatelům internetu i institucím a v neposlední řadě umožňuje používání a věrohodnost kryptoměn. Je zřejmé, že narušení spolehlivého fungování těchto algoritmů představuje značné bezpečnostní a ekonomické riziko[1].

Tato kapitola shrnuje vlastnosti prvočísel a demonstruje jejich význam pro asymetrickou kryptografii na několika příkladech šifrovacích algoritmů.

1.1 Vlastnosti prvočísel

Prvočíslo lze definovat jako přirozené číslo p větší než 1, které má pouze dva dělitele: 1 a p. Jinak řečeno, platí[2]:

$$\forall a \in \mathbb{Z}^+ \backslash \{1; p\} : p \in \mathbb{Z}^+ \backslash \{1\}, GCD(p, a) = 1 \tag{1.1}$$

Jednou z vlastností prvočísel relevantní pro tuto práci je hustota výskytu prvočísel v množině přirozených čísel. Ta se dá odhadnout vzhledem k nějakému velkému číslu x pomocí aproximace[1]:

$$\pi(x) \approx \frac{x}{\ln(x)} \tag{1.2}$$

Další důležitou vlastností prvočísel je, že Eulerova funkce jakéhokoliv prvočísla p je rovna $p-1$. Tato vlastnost se uplatní mimo jiné v podkapitole o RSA. Eulerova funkce je multiplikativní, proto je snadné vypočítat její hodnotu i pro vysoká čísla, jsou-li známé všechny jejich prvočinitele. Jestliže jsou například p, q a r prvočísla a $n = pqr$, pak platí[3]:

$$\begin{aligned} \Phi(p) &= p - 1 \\ \Phi(n) &= \Phi(p)\Phi(q)\Phi(r) \\ \Phi(n) &= (p-1)(q-1)(r-1) \end{aligned} \tag{1.3}$$

Pro kryptografii mají význam nejen obecná prvočísla, ale i některé jejich specifické skupiny se zvláštními vlastnostmi[4]:

- Mersennova prvočísla
  Tato skupina se značí M_p a patří sem čísla, pro která platí: $M_p = 2^p - 1$, kde p je prvočíslo
- Sophie-Germain prvočísla.
  S označením SG_p, platí pro ně: $p = 2SG_p + 1$, kde p je prvočíslo.
- Bezpečná prvočísla
  Značí se B_p a platí: $B_p = 2p + 1$, kde p je prvočíslo.

1.2 Algoritmy využívající prvočísla

Spolehlivost mnoha dnešních kryptografických nástrojů – zejména asymetrických – se neobejde bez používání bezpečných prvočísel. Symetrické a asymetrické algoritmy mají různé přednosti i úskalí, obojí způsob šifrování má v kryptografii své využití.

Symetrické jsou rychlejší, ale používají tajné klíče, které je potřeba přenést mezi stranami, což je problém, který je zpravidla řešen použitím asymetrické šifry. Symetrické šifry mají svá pravidla ohledně délky a vlastností klíčů, avšak na rozdíl od asymetrické kryptografie nevyžadují nutně prvočísla. Jejich bezpečnost stojí na tom, že heslo zůstane před útočníkem skryto, nikoliv na výpočetní náročnosti některých operací, které budou rozebrány v následujícím seznámení s konkrétními šiframi.

Asymetrická kryptografie používá veřejné a soukromé klíče, které jsou matematicky propojeny. Díky tomu není nutné přenášet klíče jinými kanály než zašifrovanou zprávu (například je doručovat fyzicky). Tento způsob s sebou ale přináší nebezpečí, že útočník se znalostí veřejného klíče dokáže soukromý klíč vypočítat.

Kryptografie se spoléhá na fakt, že tento matematický problém je příliš složitý na to, aby se útočníkovi vyplatilo šifru prolamovat. Nejčastěji se využívá problém diskrétního logaritmu, faktorizace velkých čísel a eliptické křivky. Následuje několik významných příkladů asymetrických šifer[1, 4].

1.2.1 Diffie-Hellmanův protokol

Algoritmus publikovaný v roce 1976, jehož autory jsou Whitfield Diffie a Martin Hellman, funguje na tomto principu:

Dvě komunikující entity – Alice a Bob – si zvolí prvočíslo p a číslo g, které je generátorem multiplikativní grupy $\mathbb{Z}_\mathrm{p}$. To znamená, že pro všechny prvky grupy existuje nějaké x pro které platí: $g^x \bmod p \in \mathbb{Z}_\mathrm{p}$.

Obě komunikující strany si následně z této grupy vyberou svůj soukromý klíč. Alice si vybere soukromý klíč a z množiny $\{2, 3, \ldots p-2\}$ a vypočítá svůj veřejný klíč $A = g^a \bmod p$. Obdobně Bob spočítá vlastní veřejný klíč $B = g^b \bmod p$.

Výstupem protokolu je klíč K, který strany použijí pro symetrické šifrování další komunikace.

$$\begin{aligned} K &= B^a \bmod p \\ K &= A^b \bmod p \\ K &= g^{ab} \bmod p \end{aligned} \tag{1.4}$$

Potenciální útočník by musel vypočítat K bez znalosti a a b. Přitom by využil tohoto vztahu:

$$K = g^{ab} \bmod p \tag{1.5}$$

$$a = \log_g B \bmod p \tag{1.6}$$

$$b = \log_g A \bmod p \tag{1.7}$$

To je problém diskrétního logaritmu a jeho řešení má exponenciální složitost, tím pádem je pro vysoká čísla v přijatelném čase neřešitelný. Princip je vysvětlen pro případ, že se komunikace účastní dvě entity, algoritmus se však dá obdobně použít i pro více účastníků[5, 6].

1.2.2 ElGamal

I tato šifra, pojmenovaná podle svého tvůrce, se opírá o složitost diskrétního logaritmu. S použitím stejného značení jako u Diffie-Hellmana lze princip algoritmu ElGamal vysvětlit takto[6, 7]:

Alice a Bob si zvolí čísla q a g, soukromé klíče a a b a navzájem si pošlou veřejné klíče A a B. Při použití této šifry odesílatel při každém přenosu volí nový klíč a posílaná zpráva musí být prvkem multiplikativní grupy $\mathbb{Z}_{\mathrm{p}}$. Když Bob posílá Alici zprávu M, provede operaci $C = MK \bmod p$, kde $K = A^b \bmod p$, Alici pošle číslo C a svůj veřejný klíč. Alice následně dešifruje přenos pomocí inverzního prvku K^{-1}:

$$K = B^a \bmod p \tag{1.8}$$

$$KK^{-1} \equiv 1 \pmod{p} \tag{1.9}$$

$$M = CK^{-1} \bmod p \tag{1.10}$$

1.2.3 DSA – Algoritmus digitálního podpisu

Anglicky „Digital Signature Algorithm“, slouží k zajištění integrity a nepopiratelnosti. Když entita Alice pošle entitě Bob dokument a připojí k němu digitální podpis, má Bob jistotu, že dokument jednoznačně pochází od Alice a nebyl v průběhu doručování změněn.

Strany se nejdříve dohodnou na číslech p, q a g. Tato čísla mohou být veřejně známá a musí splňovat podmínky: p a q jsou prvočísla, pro libovolné $k \in \mathbb{Z}_{\mathrm{q}}$ platí: $p - 1 = kq \Rightarrow gq \equiv 1 \pmod{p}$.

Aby se dal podpis jednoznačně připojit k dokumentu, vstupuje do algoritmu také otisk tohoto dokumentu získaný pomocí hešovací funkce, otisk je dále značen $H(m)$. Vytváření podpisu probíhá takto:

Alice si určí soukromý klíč a a vypočítá veřejný klíč $A = g^a \bmod p$. Pak vybere náhodné číslo $k \in \mathbb{Z}_q$, tak aby $g^k \not\equiv 0 \pmod{q}$ a vypočítá:

$$r = (g^k \bmod p) \bmod q \tag{1.11}$$

$$s = k^{-1}(H(m) + ar) \bmod q \tag{1.12}$$

Alice pošle Bobovi dokument, svůj podpis – dvojici (r, s) a svůj veřejný klíč A. Bob podpis ověří těmito výpočty:

$$w = s^{-1} \bmod q$$
$$u_1 = H(m)w \bmod q$$
$$u_2 = rw \bmod q$$
$$v = (g^{u_1} A^{u_2} \bmod p) \bmod q$$

Když vypočtené v je rovno číslu r, které poslala Alice, znamená to, že podpis opravdu pochází od Alice.

Aby bylo možné se na tento algoritmus spolehnout, má být podle NIST délka p minimálně 1024 bitů a délka q minimálně 160 bitů. Je doporučeno používat dvojice o délkách: (1024, 160), (2048, 224), (3072, 256), (7680, 384) a (15360, 512)[5, 8].

1.2.4 Eliptické křivky

Eliptická křivka je algebraická struktura konstruovaná nad tělesem. Je to hladká spojitá křivka, na které je definován bod O ležící v nekonečnu. Pro účely kryptografie je možné eliptickou křivku nad reálnými čísly definovat jako skupinu souřadnic (x, y), které vyhovují například rovnici $y^2 = x^3 + ax + b$, kde $a, b, x, y \in \mathbb{R}$.

Pro kryptografické algoritmy na bázi eliptických křivek, anglicky „Elipic Curve Cryptography“, se používá skalární násobení bodů na náležících jisté křivce v konečném poli. Takové body nelze rovnou násobit, operace vyžaduje postupné sčítání. Jedná se o výpočetně obtížné problémy vedoucí k faktorizaci velkých čísel nebo k diskrétnímu logaritmu[9].

Problém diskrétního logaritmu nad eliptickou křivkou lze popsat takto: Máme-li body $Q, R \in Ep(a, b)$ a rovnici $Q = kR$, kde $k < p$, k lze snadno odvodit ze znalosti R. Avšak nalezení čísla k pouze při znalosti Q a R je výpočetně obtížné. Takové kladné celé číslo k se nazývá „Diskrétní logaritmus bodu Q vzhledem k základu R nad nad eliptickou křivkou E“[5, 10].

Výhodou eliptických křivek je, že se stejně dlouhými klíči poskytují významně vyšší úroveň bezpečnosti než ostatní algoritmy. Například RSA, které používá n délky 2048 bitů má stejnou úroveň bezpečnosti jako ECC s prvočíslem dlouhým 224 bitů. Doporučené délky prvočísel pro algoritmy založené na eliptických křivkách jsou 192, 224, 256, 384 a 512 bitů[11].

1.2.5 RSA

Šifra RSA je pojmenovaná podle svých autorů – Ronald Rivest, Adi Shamir a Max Adleman. Její algoritmus pracuje se dvěma prvočísly r a s, jejich součinem $n = rs$ a číslem $e < n$. e je nesoudělné s číslem $v = \Phi(n)$, platí tedy: $GCD(e, v) = 1$.

Složitost šifry spočívá v tom, že jsou-li r a s dostatečně velká, není snadné vypočítat v pouze při znalosti n. Jedná se tedy o problém faktorizace velkých čísel.

Číslo e je použito jako veřejný klíč $Pk(e, n)$. Je vhodné použít $e = 2^x + 1$, často se používá $e = 65537 (= 2^{16} + 1)$. Jako soukromý klíč $Sk(d, n)$ se používá inverzní prvek k e.

$$d = e^{-1} \Rightarrow de \equiv 1 \pmod{v} \tag{1.13}$$

Šifrování: (obě strany znají r, s, e, jsou schopné vypočítat d)

$$m \longrightarrow c = m^e \bmod n$$

Dešifrování:

$$c \longrightarrow m = c^d \bmod n = m^{ed} \bmod n = m^1 \bmod n$$

RSA se používá jednak k utajení zprávy m a jednak k elektronickým podpisům. Při výměně tajné zprávy pošle Alice Bobovi zašifrovanou zprávu c, viz výše, a veřejný klíč $Sk(e, n)$. Při podepisování Alice pošle Bobovi zprávu m a podpis $Sig_{\mathrm{m}} = m^d \bmod n$. Bob podpis ověří výpočtem $m = Sig_{\mathrm{m}}^e \bmod n$. Když se m shoduje se zprávou, kterou Alice poslala s podpisem, má Bob jistotu, že zpráva pochází od Alice[6, 7, 12].

Prolomení RSA se slabými klíči

Následující příklad ukazuje, proč je důležité, aby vstupní prvočísla měla určitou minimální velikost (bitovou délku). V tomto příkladu se útočník snaží prolomit šifru RSA, která používá příliš krátká r a s. Útočník zachytil zprávu c a zná veřejný klíč $Sk(e, n)$. Snaží se zjistit $v (= \Phi(n))$, aby mohl vypočítat soukromý klíč d a zprávu jím dešifrovat.

Pro výpočet může útočník použít Fermatovu faktorizaci:

$$rs = n = x^2 - y^2 = (x - y)(x + y) \Rightarrow r = x - y; s = x + y$$

Útočník hledá x a y, tak aby:

$$x^2 = y^2 - n \tag{1.14}$$

Algoritmus pro nalezení x a y nejdříve položí $x_0 = [\sqrt{n}]$ (nejvyšší celé číslo menší než $\sqrt{n}$). Poté zkouší, zda rovnice platí pro různá x_{i} z množiny

$X = \{x_0 + 1; x_0 + 2; \dots\}$ tak dlouho, dokud nenarazí na x_i a $y_\mathrm{i} = [\sqrt{|n - x_\mathrm{i}^2|}]$, pro která platí rovnice 1.14. To nastane v situaci, kdy odmocnia z $n - x_\mathrm{i}^2$ bude rovnou celé číslo, takže bude platit $[\sqrt{|n - x_\mathrm{i}^2|}] = \sqrt{|n - x_\mathrm{i}^2|}$. Když útočník nalezne vhodné x a y, je už snadné vypočítat:

$$r = x - y \tag{1.15}$$

$$s = x + y \tag{1.16}$$

$$v = (s - 1)(r - 1) \tag{1.17}$$

Tento způsob prolamování je obzvlášť efektivní, je-li rozdíl mezi s a r relativně malý.

Příklad: Alice a Bob si zvolili 8bitová prvočísla:
$r = 151; s = 199 \Rightarrow n = 151 \times 199 = 30049$.
Veřejný klíč v tomto příkladu bude $e = 17 (= 2^4 + 1)$.

Alice posílá Bobovi zprávu $m = 12345$, po zašifrování bude $c = m^e \bmod n$, tedy $c = 12345^{17} \bmod 30049 = 23215$.

Útočník bude znát $c = 23215, e = 17$ a $n = 30049$.
Postup faktorizace:

$$x_0 = [\sqrt{n}] = [\sqrt{30049}] = 173$$

1. krok:

$$\begin{aligned}
x_1 &= 173 + 1 = 174 \\
y_1 &= [\sqrt{|n - x_1^2|}] \\
y_1 &= [\sqrt{|30049 - 174^2|}] \\
y_1 &= [\sqrt{227}] \\
y_1 &= 15 \\
x_1^2 - y_1^2 &= 174^2 - 15^2 \\
x_1^2 - y_1^2 &= 30051 \\
30051 &\neq n
\end{aligned}$$

2. krok:

$$\begin{aligned} x_2 &= 174 + 1 = 175 \\ y_2 &= [\sqrt{|n - x_2^2|}] \\ y_2 &= [\sqrt{|30049 - 175^2|}] \\ y_2 &= [\sqrt{576}] \\ y_2 &= 24 \\ x_2^2 - y_2^2 &= 175^2 - 24^2 \\ x_2^2 - y_2^2 &= 30049 \\ 30049 &= n \end{aligned}$$

Správná x a y byla zjištěna již po dvou krocích.

$$\begin{aligned} r = x - y &= 175 - 24 = 151 \\ s = x + y &= 175 + 24 = 199 \\ v = \Phi(n) &= (r-1)(s-1) \\ v &= 29700 \end{aligned}$$

Následně útočníkovi stačí najít soukromý klíč d:

$$\begin{aligned} de &\equiv 1 \pmod{v} \\ 17d &\equiv 1 \pmod{29700} \\ d &= 27953 \end{aligned}$$

Nyní může útočník dešifrovat zprávu:

$$\begin{aligned} m &= c^d \bmod n \\ m &= 23215^{27953} \bmod 30049 \\ m &= 12345 \end{aligned}$$

Protože byla použita pouze 8bitová prvočísla, byl výpočet časově nenáročný. Ve skutečnosti se pro RSA používají čísla typicky o délce 2048 až 4096 bitů. Díky tomu, že tak vysoká čísla současné počítače nedokážou faktorizovat dostatečně rychle, může tento algoritmus spolehlivě utajit důvěrnou komunikaci. Pro n dlouhé 2048 bitů je potřeba, aby s a r měla délku přibližně 512 bitů[13].

1.3 Složitost

Za spolehlivou (bezpečnou) šifru považujeme takovou, kterou se potenciálnímu útočníkovi nevyplatí prolamovat. V této podkapitole bude vysvětleno, proč nemá smysl se pokoušet prolamovat šifry s bezpečnými vstupními čísly, jak je uvedeno výše.

Algoritmy se dají rozdělit podle časové nebo paměťové složitosti. Paměťová složitost je důležitá, když si útočník vymezuje zdroje k útoku. Bezpečnost asymetrických šifer se ale spoléhá především na jejich časovou složitost. Je-li časová složitost exponenciální $O(2^n)$, faktoriální $O(n!)$ nebo dvojnásobně exponenciální $O(n^n)$, je snaha o prolomení poměrně zbytečná již při vstupu $n = 100$.

Složitost faktorizace a výpočtu diskrétního logaritmu závisí na výběru konkrétních algoritmů. Obecně se dá označit za subexponenciální. Při použití dnes dostupných počítačů by prolomení RSA s 2048bitovými klíči (délka, kterou doporučuje NIST) trvalo přibližně 3×10^{14} let.

Kdyby byl sestrojen nedeterministický Turingův stroj, tento výpočet by mu trval významně kratší dobu. Ten však sestrojen nebyl a není jisté, zda se vůbec sestrojit dá[13, 14, 15].

2 Generování prvočísel

Kromě šifrovacích algoritmů jsou pro kryptografii zásadní hodnoty na vstupu těchto operací. Na uvedených příkladech je vidět, že těmito vstupy jsou klíče a číselné hodnoty přenášených zpráv. Kromě nich kryptografie často pracuje s výplněmi a solí.

Čísla určená k těmto účelům je také potřeba před potenciálními útočníky tajit. Zpravidla nedochází k jejich přenosu mezi účastníky komunikace, proto stačí zajistit, aby útočník nemohl předpovědět, jaké číslo si daná entita zvolí. Proto není dobrý nápad tato čísla používat opakovaně, nebo někde zveřejňovat „databáze vhodných klíčů“. Je pouze doporučená délka – rozsah ve kterém je vhodné čísla volit – a každá entita si čísla vygeneruje sama. K tomu může použít náhodně vygenerovanou posloupnost čísel[1, 16].

2.1 Ideální generátor

Na náhodně generované posloupnosti čísel jsou kladeny následující nároky: Nepredikovatelnost, rovnoměrné rozložení, nezávislost a vysoká rychlost. Ideální generátor by měl všechny tyto požadavky dodržet. V případě, že útočník správně odhadne posloupnost vygenerovaných hodnot použitých k šifrování, samotný šifrovací algoritmus bude představovat pouze zdržení útoku, nikoliv dostatečnou překážku[16].

2.1.1 Entropie

Generátor je tím kvalitnější, čím větší má entropii. Entropie je v kontextu kryptografie definována jako míra náhodnosti. Entropie generátoru je maximální, jsou všechny posloupnosti určité délky generovány se stejnou pravděpodobností. Když je naopak možné s jistotou předpovědět generovanou posloupnost, je entropie generátoru nulová.

Za předpokladu, že hodnoty z množiny $X = \{x_1, x_2, \dots x_\mathrm{n}\}$ jsou generovány s pravděpodobnostmi $p_1, p_2, \dots p_\mathrm{n}$, je entropie:

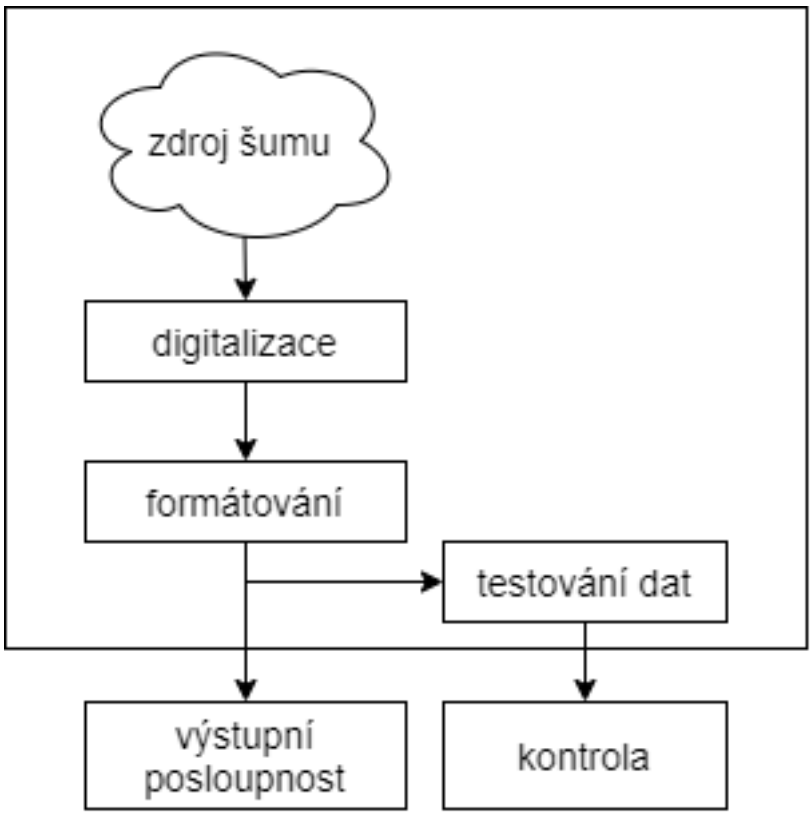
$$H(X) = -\sum_{i=1}^{n} p_\mathrm{i} log_2 p_\mathrm{i} \tag{2.1}$$

Nelze jednoznačně změřit, zda jsou generátory naprosto náhodné, existují však metody, které mohou detekovat jevy svědčící o nedostatečné entropii. NIST publikuje několik způsobů, jak se dá otestovat, zda generátor poskytuje dostatečnou entropii. Několik testů, kterými se dá odhalit nedostatek generátoru:

- frekvenční test – porovnání počtu jedniček a nul,
- „runs“ test – testuje počet a délku řetězců po sobě jdoucích stejných bitů,

- test hodností matic – hodnost disjunktních podmatic (odhalení lineární závislosti),
- spektrální test – diskrétní Fourierova transformace.

Generátory můžeme rozdělit podle toho, odkud pochází náhodnost posloupností. Zdroj náhody může být přímo v přírodě, nebo vytvořený uměle použitím algoritmu[16, 17].



Obr. 2.1: Model zdroje entropie.

2.2 Fyzikální generátory (TRNG)

Generují čísla na základě měření fyzikálních veličin. Tyto generátory se označují za skutečně náhodné, v angličtině „True Random Number Generators“, protože náhodnost generované posloupnosti pochází z přírodních jevů, které nelze předvídat. Zároveň je vhodné poznamenat, že ne každý fyzikální generátor je spolehlivě náhodný. Záleží na tom, jestli je pozorovaný jev skutečně nepředvídatelný a jestli stroj interpretující naměřené hodnoty nezasahuje do jejich rozložení.

Naměřená data je zpravidla potřeba upravit a převést na žádoucí formu posloupnosti čísel, která se dál zpracuje. Je třeba ověřit, zda vybraný způsob měření nebo

specifické podmínky nezasáhly do entropie generování. Po převedení dat do požadovaného formátu je na místě takzvaný „health test“, který ověří, zda mají naměřená data požadované vlastnosti.

Příklady fyzikálních jevů, které může generátor použít:

- akustický šum,
- tepelný šum,
- atmosférický šum,
- záření na pozadí/kosmické záření,
- radioaktivní rozpad,
- chování některých elementárních částic, například fotonů procházejících polopropustným zrcadlem,
- události na hardwaru počítače – pohyby myši, frekvence úderů do klávesnice atd.

Nevýhodou skutečně náhodných generátorů je velké množství vstupních dat, s ním související časová náročnost konverze na čísla a neopakovatelnost sekvence. Tato vlastnost sice svědčí o vítané míře náhodnosti, avšak v situaci, kdy je potřeba testovat program s náhodným čísly na vstupu, může být překážkou[17, 18].

2.3 Algoritmické generátory (PRNG)

Na rozdíl od přírodních jevů se počítače chovají deterministicky. Jsou sestrojeny takovým způsobem, že každý krok algoritmu vede jednoznačně z jednoho stavu do druhého, a pro konkrétní nezměněný vstup má určitý program vždy stejný výstup.

Přesto jsou počítače schopné generovat řady čísel, které sdílí statistické vlastnosti náhodných posloupností a mají dostatečnou míru entropie pro potřeby kryptografie. Takto vzniklé posloupnosti se nazývají pseudonáhodné. Podle toho se algoritmické generátory někdy označují pseudonáhodné, anglicky „Pseudorandom Number Generators“.

Tyto generátory potřebují vstupní hodnotu, tzv. seed. Na ni aplikují algoritmus, který v několika cyklech vygeneruje posloupnost čísel. Výsledná posloupnost sice není náhodná, protože jednoznačně vychází z algoritmu a seedu, ale když útočník nezná použitý seed, nedokáže předpovídat následující číslo posloupnosti.

Tyto algoritmy fungují na principu blokové šifry v režimu čítače, proudové šifry, nebo jednosměrné funkce „hash“[16, 18].

2.3.1 Von Neumanova metoda (Middle-square method)

Typickým příkladem generátoru pro vytváření pseudonáhodných posloupností je algoritmus, který představil John von Neumann v roce 1949. Postupuje tak, že z

množiny n-ciferných čísel je vybráno jedno číslo x_0 (seed), které je umocněno na druhou. Výsledné x_0^2 má být zapsáno právě $2n$ ciframi. Pokud je kratší, je potřeba zleva doplnit dostatečný počet nul. Následující číslo posloupnosti x_1 je získáno výběrem n prostředních cifer z x_0^2, odtud pojmenování „metoda prostředních řádů“.

Na obrázku 2.2 je znázorněno, jak vzniká posloupnost při aplikaci této metody na seed=1234. Každý sloupec představuje průběh jednoho cyklu algoritmu. Doplnění nulami je zvýrazněno modře a výběr středních řádů je označen červenou barvou. Výsledná posloupnost je: $\{1234, 5227, 3215, 3362, 3030, 1809, 2724, 4201, 6484, 422, \dots\}$.

x:	1234	5227	3215	3362	3030	1809	2724	4201	6484
x^2:	1522756	27321529	10336225	11303044	9180900	3272481	7420176	17648401	42042256
padded:	01522756	27321529	10336225	11303044	09180900	03272481	07420176	17648401	42042256

Obr. 2.2: Posloupnost generovaná von Neumannovou metodou se seedem 1234.

Pro algoritmy tohoto typu platí, že po několika cyklech se v posloupnosti objeví seed. V tomto bodě už výstup generátoru přestane být náhodný, protože se celá posloupnost opakuje od začátku a je tedy stoprocentně předvídatelná. K tomuto opakování zpravidla dochází po časové periodě, která závisí na délce čísel. Zároveň pro takový algoritmus platí, že ve vygenerované posloupnosti (jestliže generování skončí dříve než se seed opakuje) se každé číslo objeví maximálně jednou[16, 19].

2.3.2 Kongruenční generátor

Dalším z algoritmů, kterými se dá generovat pseudonáhodná posloupnost, je kongruenční generátor. Obecně se jedná o cyklus, ve kterém každá hodnota posloupnosti je výsledkem funkce předchozích hodnot. Podle toho, jaká funkce je použita, rozlišujeme několik typů, například[16]:

- lineární kongruence: $x_{n+1} = (ax_n + b) \bmod m$,
- multiplikativní kongruence: $x_{n+1} = (a_n x_n) \bmod m$,
- aditivní kongruence: $x_{n+1} = (x_n + x_{n-1}) \bmod m$,
- různé kombinace předchozích.

2.3.3 Blum-Blum-Shrub

$$x_{n+1} = x_n^2 \bmod m \tag{2.2}$$

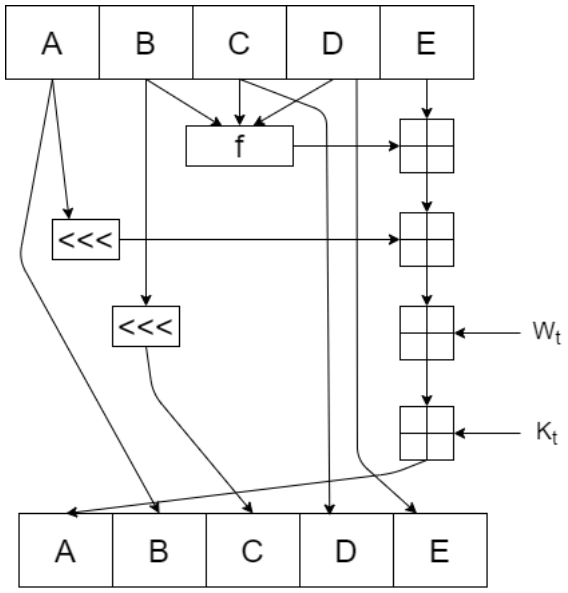
Postup této metody se podobá von Neumannovu algoritmu, s tím rozdílem, že místo výběru cifer sníží řád čísla pomocí operace modulo. Jako modulární operátor je

zvolen součin prvočísel $pq = m$. Jako důkaz bezpečnosti tohoto algoritmu se používá výpočetní náročnost faktorizace velkých celých čísel. Při znalosti p a q je možné získat i-tý prvek posloupnosti bez nutnosti generování předchozích prvků[16].

$$x_{\mathrm{i}} = (x_0^{2^i \bmod \Phi(n)}) \bmod m \tag{2.3}$$

2.3.4 Linux PRNG

Tato metoda využívá data z hardwaru počítače, jako jsou pohyby myši, údery do klávesnice a čas uplynulý od posledního načtení disku. Tato data zpracuje pomocí funkce SHA-1. To je hešovací funkce která vstup rozdělí na bloky o délce 512 bitů, každý blok pak na 32bitová slova A, B, C, D a E. Následně provede 80 rund podle schématu na obrázku 2.3. f značí proměnnou nelineární funkci, K_t je konstanta a W_t je proměnné slovo. Symbol <<< označuje bitovou rotaci a + znamená provedení operace XOR[16, 20].



Obr. 2.3: Schéma rundy hashovacího algoritmu SHA-1.

2.4 Smíšené generátory

Jak už napovídá název, smíšené generátory spojují dvě předchozí metody generování. Zpravidla tím způsobem, že použijí naměřenou hodnotu přírodního jevu jako vstup (seed) generovacího algoritmu[16].

2.5 Testování prvočíselnosti

Tato podkapitola se věnuje způsobům, jak vybrat z vygenerovaných čísel prvočísla. Protože posloupnost na výstupu generátoru je náhodná nebo pseudonáhodná, je zřejmé, že zdaleka ne všechna čísla v ní jsou prvočísly. K výběru prvočísel z posloupnosti přirozených čísel se používají testy prvočíselnosti.

Testy prvočíselnosti jsou dvojího druhu – skutečné a pravděpodobnostní. Použití pravděpodobnostního testu nezaručuje stoprocentní spolehlivost. Pravděpodobnost, že test označí složené číslo za prvočíslo, je ale tak nízká, že se i přesto v praxi využívají. Vůči skutečným testům mají pravděpodobnostní testy výhodu, že nekladou příliš vysoké nároky na výkon[4, 21].

2.5.1 Fermatův test

Využívá malou Fermatovu větu: Jestliže n je prvočíslo a a je kladné celé číslo z intervalu $(0; n)$ pak $a^{n-1} \equiv 1 \pmod{n}$.

V průběhu testu se volí náhodná čísla a_{i} a dosazují se do rovnice. Dokud rovnice platí, může se jednat o prvočíslo. Jakmile nastane případ nerovnosti: $a_{\mathrm{i}}^{n-1} \not\equiv 1 \pmod{n}$ pak n není prvočíslo a a_{i} je tzv. Fermatův svědek složenosti.

Fermatův test se příliš nevyužívá. I po vyzkoušení všech a je možné, že kongruence bude platit, přestože n nebude prvočíslem. V takovém případě je n Carmichaelovo číslo (pseudoprvočíslo) a a_{i} je tzv. Fermatův lhář[4, 22].

2.5.2 Miller-Rabinův test

Také se jedná o pravděpodobnostní test, ale je spolehlivější než Fermatův. Postup:

1. n se vyjádří ve tvaru $n = 2^s r + 1$ a najdou se čísla r a s, pro která tato rovnice platí (např. $r = n - 1 \bmod 2$). r bude liché číslo.
2. Vybere se náhodné celé číslo a z intervalu $(0; n)$.
3. Je-li n prvočíslem, musí platit $a^r \equiv 1 \pmod{n} \vee a^{2^i r} \equiv n - 1 \pmod{n}$ pro nějaké $i \in (-1; s)$.
   Jestliže pro libovolné a neplatí ani jedna kongruence pro žádné i, pak je n složené.
4. Jestli alespoň jedna kongruence platí, test je možné opakovat pro jiné a pro vyšší jistotu.

U tohoto testu se dá snadno zjistit, s jakou pravděpodobností se mýlí. Jestliže Miller-Rabinův test proběhne x-krát (pro $a \in \{a_1, a_2, \ldots a_{\mathrm{x}}\}$), označí složené číslo za prvočíslo s pravděpodobností $\frac{1}{4^x}$[4, 22].

2.5.3 Lucas-Lehmerův test

Tento test je na rozdíl od předchozích skutečným testem prvočíselnosti. Jestliže jím číslo projde, jedná se zaručeně o prvočíslo. Má však i jisté nevýhody: Je náročnější něž pravděpodobnostní testy a je navržen tak, aby identifikoval pouze Mersennova prvočísla – prvočísla tvaru $p = 2^s - 1$, kde s je také prvočíslo.

Jelikož ne každé prvočíslo je Mersennovo prvočíslo, test může falešně označit prvočíslo za složené číslo. Při rozhodování, zda aplikovat například Miller-Rabinův nebo Lucas-Lehmerův test, je třeba (kromě náročnosti výpočtu) porovnat význam falešně pozitivních a falešně negativních výsledků.

Testované n se vyjádří ve tvaru $n = 2^s - 1$. Test probíhá tak, že se spočítá řada pomocných čísel $u_0, u_1, \ldots u_{s-2}$. Řada začíná číslem $u_0 = 4$ a každé další číslo se vypočte z předchozího jako $u_k = (u_{k-1}^2 - 2) \mod n$. Pokud $u_{s-2} = 0$, pak lze s jistotou říci, že n je Mersennovo prvočíslo[4, 21].

3 Programovatelné obvody

Generátor je navržen na platformě FPGA v jazyce VHDL. Do prostředí Xilinx Vivado, které umožňuje implementaci a simulaci návrhu, je přistupováno pomocí připojení ke vzdálené ploše.

Tato kapitola stručně popisuje platformu FPGA, jazyk VHDL, některé jeho prvky užité v implementaci generátoru a některé v praxi implementované možnosti náhodného generování čísel.

3.1 FPGA

Field Programable Gate Arryay – programovatelné hradlové pole – je typ elektrického obvodu jehož funkci lze pomocí návrhu specifikovat mimo výrobní linku. Obvody se dají rozlišit na dvě kategorie podle toho, kam se ukládají jejich konfigurační informace. FPGA s volatilní konfigurací je mají uložené v paměťových buňkách SRAM. Oproti tomu obvody s nevolatilní konfigurací přechovávají tato data jinde, například ve flash paměti.

Blokovou strukturu FPGA tvoří pole vzájemně propojených konfigurovatelných logických bloků (CLB), obklopené vstupně/výstupními bloky. Jejich propojení je horizontální a vertikální a také se dá definovat pomocí návrhového jazyka. V poli se mohou objevit i jiné bloky se specifickými funkcemi, jako jsou multiplexory použité jako spínače. Konfigurovatelné logické bloky obsahují generátor logické funkce a klopný obvod. FPGA obsahují kromě univerzálních logických buněk také přídavné prvky se speciálním zaměřením pro potřeby často implementovaných zapojení[23].

3.2 VHDL

VHDL (VHSIC Hardware Description Language) je jedním z hlavních jazyků, které se dají použít pro návrh hradlových polí. Původně byl vytvořen pro dokumentaci zákaznických obvodů (již „hotových“ obvodů bez možnosti návrhu) a dá se použít i pro návrh analogových obvodů. Jeho dvě hlavní funkce jsou simulace obvodů a dokumentace instrukcí pro výrobu obvodů. Dokáže popsat obvody na algoritmické, logické i hradlové úrovni.

Zdrojové kódy VHDL jsou typicky sestaveny z popisů jednotlivých komponent. Při implementaci se objeví (kromě deklarace knihoven bez kterých se neobejde snad žádný pokročilejší programovací jazyk) entita – struktura pro identifikaci hardwarové součástky na základě vstupů a výstupů, a architektura – definice chování komponenty[23, 24].

3.2.1 Implementace elementárních operací

VHDL se příliš nehodí k návrhu komponent se složitými výpočetními operacemi. Pro tento jazyk je přirozené zpracovávat přímo elektrické signály a provádět s nimi běžné bitové operace – and, or, sčítání, odčítání, bitový posun a rotace. Složitější operace – násobení, mocniny, modulo, atd. – lze použít pro vybrané datové typy a v některých případech je nutné je „obejít“ použitím cyklů a kombinací jednodušších operací[24].

3.2.2 Typy signálů

Vstupy a výstupy číslicových systémů jsou v dnešní době předávány binárně. Proto je vhodné při programování hardwarové realizace použít typ *bit* který nabývá hodnot 0 a 1. V jazyce VHDL lze použít typy jako *boolean*, *integer* a *string* avšak jeho specialitou jsou typy zapisované přímo jako bity, respektive bitové vektory – *std_logic* a *std_logic_vector*. Ty se dají převést na typy *signed* a *unsigned* (podle toho, zda je potřeba počítat se znaménkem, nebo má daný signál nést pouze kladná čísla)[24].

3.3 Doposud známé implementace generátorů

Tato podkapitola uvádí několik příkladů zveřejněných implementací generování náhodných řad bitů na platformě FPGA. Některé navržené generátory jsou určeny pro generování nízkých čísel a velké množství jich neobsahuje specifický prvek testování prvočíselnosti. Následující výčet uvádí různé zdroje entropie použité v implementacích TRNG.

3.3.1 Tepelný šum

Elektronický šum, v literatuře popsaný jako Johnson–Nyquist noise, vzniká ve vodiči v důsledku zahřátí a není ovlivněn napětím. Tento šum se dá změřit v určitém okamžiku, naměřená hodnota se pak použije jako referenční a entropie bude pocházet z rozdílu referenční hodnoty od aktuálně měřeného šumu[25].

3.3.2 Jitter

Zařízení nazvané Ring oscilator používá jako zdroj entropie jitter. Je tvořeno smyčkou, ze které vystupuje signál *true* nebo $false$, v závislosti na aktuální výši proměnlivého napětí. Tak se dá oscilátor použít ke generování náhodných bitů, a ty pak mohou tvořit náhodný vektor[26].

3.3.3 Kvantový efekt

Speciální typ fyzikálního generátoru je QRNG. Je založen na detekci fotonu po průchodu děličem paprsků. Foton v takovém případě může jít jednou ze dvou možných cest. Na základě toho, kterou cestou projde, je na výstupu generátoru 0 nebo 1[25].

3.3.4 Metastabilita

Tento zdroj entropie využívá náhodné prvky v chování obvykle již přítomných součástí obvodu. V závislosti na verzi obvodu lze použít metastabilní operace klopných obvodů, čas posledního průchodu oscilátorem, nebo kolize při ukládání dat do paměti Block Memory[25].

4 Porovnání a výběr metod

V této kapitole jsou výše představené metody generování a metody testování mezi sebou porovnány. Jsou diskutovány jejich výhody a nevýhody a následně je navrženo několik možných kombinací pro implementaci generátoru prvočísel. Kombinace vyhodnocená jako nejvhodnější je v následující kapitole převedena do formy kódu v jazyce VHDL.

4.1 Porovnání nastudovaných metod

Následující sekce se věnuje srovnání algoritmů uvedených ve druhé kapitole. Generátor bude cyklicky postupovat ve dvou hlavních krocích:

1. vygenerování náhodného čísla,
2. otestování, zda se jedná o prvočíslo.

4.1.1 Srovnání metod generování

Co se týká metod generování, nabízejí se fyzikální, algoritmické a smíšené generátory. Fyzikální mají obecně nevýhodu náročného zpracovávání velkého množství dat, zároveň však poskytují skutečnou náhodnost. Posloupnosti generované použitím TRNG jsou jedinečné a nepředvídatelné.

S přihlédnutím k míře náhodnosti, jakou typicky používané jevy poskytují, nemá příliš smysl srovnávat jejich kvalitu. Výběr jevu, ze kterého TRNG čerpají entropii se zpravidla řídí tím, kterou veličinu lze na daném obvodu nejefektivněji měřit a zpracovávat.

Jak už je uvedeno v teoretické části práce, algoritmické generátory produkují posloupnosti v závislosti na jistých pravidlech. Jsou-li vstupní hodnoty udrženy v tajnosti, i posloupnosti generované PRNG splňují podmínku nepředvídatelnosti, dokud nedojde k zacyklení algoritmu. Je proto třeba zajistit, aby opakování seedu nezpůsobilo ztrátu nepředvídatelnosti. Výhodou je, že při využití PRNG není nutné provádět operace měření náhodného jevu a převádění tohoto jevu na číslo v požadovaném formátu.

Další možností je smíšený generátor. Použití fyzikálně vygenerovaného seedu vnese prvek skutečné entropie z přírodního jevu do časově efektivního algoritmu. V tomto případě opakování posloupnosti stále hrozí nezávisle na tom, odkud seed pochází. Tomu může zabránit kontrola posloupnosti v průběhu procesu generování. Kdyby taková kontrola odhalila opakování výstupních hodnot, algoritmus by byl přerušen a spuštěn znova s novým seedem. V případě, že generátor bude pracovat

s čísly v rozsahu odpovídajícím bezpečným prvočíslům ale taková situace není pravděpodobná. Čím větší je rozsah výstupní posloupnosti (čím více má číslic), tím méně je pravděpodobné, že se v omezeném počtu výstupních čísel vyskytne právě vstupní hodnota.

Ve volbě mezi PRNG a TRNG hrají roli specifické požadavky na generátor. Tak jako s většinou ideálních modelů se skutečná realizace může teoretickému vzoru pouze blížit, a často lze vyhovět jen jedné vlastnosti na úkor druhé.

Generátor implementovaný v této práci má kromě generování náhodné posloupnosti navíc obsahovat test prvočíselnosti. Ten bude využívat složité matematické operace, proto se spokojí s algoritmickým generátorem a ušetří čas potřebný k testování. Pro účely práce bude stačit využití některého pseudonáhodného algoritmu, případně kombinace dvou PRNG metod pro oddálení zacyklení řady. Kdyby se měl generátor dlouhodobě využívat, je vhodné zvážit přidání TRNG prvku pro předání nového seedu na vstup PRNG algoritmu.

Výběr PRNG algoritmu

Chceme-li srovnávat výkonnost von Neumanovy, Blum-Blum-Shrub a kongruenční metody, je třeba se zaměřit na rychlost jednotlivých matematických operací, ze kterých jsou tyto algoritmy složeny. Bitová složitost základních operací v množině přirozených čísel je uvedena v tabulce 4.1[10].

operace	bitová složitost
sčítání	$O(\log_2 n)$
odčítání	$O(\log_2 n)$
násobení	$O((\log_2 n)^2)$
dělení	$O((\log_2 n)^2)$
modulo	$O((\log_2 n)^2)$

Tab. 4.1: Bitová složitost operací v $\mathbb{Z}$.

algoritmus	operace	celková složitost
von Neumann	2. mocnina	$O((\log_2 n)^2)$
aditivní kongruence	sčítání, modulo	$O((\log_2 n) + (\log_2 n)^2)$
multiplikativní kongruence	násobení, modulo	$O((\log_2 n)^4)$
Blum-Blum-Shrub	2. mocnina, modulo	$O((\log_2 n)^4)$

Tab. 4.2: Odhadovaná bitová složitost generovacích algoritmů.

Po odhadu složitosti na základě dostupných dat se zdá, že nejrychlejší z uvedených by mohla být metoda středních řádů a poté metoda aditivní kongruence. Jako nejvíce časově náročné se jeví Blum-Blum-Shrub a multiplikativní kongruence, respektive lineární kongruence obecně. V tabulce 4.2 jsou seřazeny algoritmy podle odhadované složitosti.

4.1.2 Srovnání metod testování

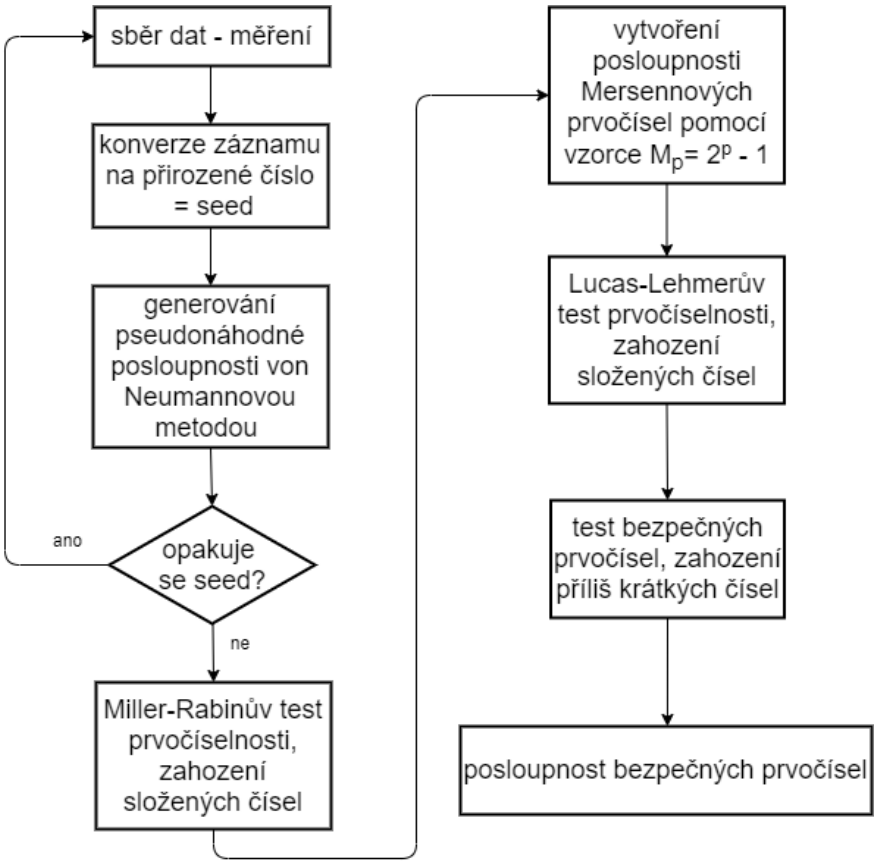
Fermatův test je z uvedených nejjednodušší a zároveň nejméně spolehlivý. Citovaná literatura jej uvádí spíše jen jako příklad, nikoli jako metodu, která by měla v současné kryptografii velký význam.

U Miller-Rabinova testu je možné spočítat pravděpodobnost omylu a na základě ní regulovat počet opakování testu pro každé číslo posloupnosti, aby byla míra pravděpodobnosti dostačující. Jestliže není potřeba mít absolutní jistotu, že vygenerovaná posloupnost je složená pouze z prvočísel, Miller-Rabinův test se jeví jako nejvhodnější možnost.

Lucas-Lehmerův test sice poskytuje jistotu, je ale ze všech nejsložitější a zahazuje mnoho prvočísel, která by se jinak dala použít. Je otázkou, zda je žádoucí, aby posloupnost obsahovala pouze Mersennova prvočísla.

V případě, že je posloupnost Mersennových prvočísel vítaným výstupem generátoru, dal by se Lucas-Lehmerův test využít tímto způsobem:

1. Vygeneruje se posloupnost poměrně krátkých čísel $a_1, a_2, \ldots a_\mathrm{k}$.
   Následující operace budou méně časově náročné, než kdyby se od začátku jednalo o posloupnost například 1024bitových čísel.
2. Posloupnost se otestuje Miller-Rabinovým testem, vyberou se prvočísla a zbytek čísel se zahodí. Vznikne nová posloupnost $b_1, b_2, \ldots b_\mathrm{m}$.
3. S každým číslem posloupnosti se provede operace $x = 2^b - 1$.
   V tomto bodě se jednak čísla významně zvýší (proto předchozí operace mohou probíhat rychleji s nízkými vstupy) a jednak do dalšího kroku vstupuje posloupnost hodnot, pro které platí alespoň jeden požadavek na Mersennovo prvočíslo, tedy že b je (pravděpodobně) prvočíslem, a zbývá už jen otestovat prvočíselnost x. Tím pádem nedojde k velkým ztrátám složitě získaných čísel x, pro která třeba ani neplatí rovnost s $2^b - 1$.
4. Lucas-Lehmerův test z posloupnosti $x_1, x_2, \ldots x_\mathrm{n}$ vybere Mersennova prvočísla.



Obr. 4.1: Schéma generátoru Mersennových prvočísel s využitím postupu „kombinace MR + LL“.

Ve srovnání s použitím Lucas-Lehmerova testu na posloupnost vysokých čísel, která jsou z velké části smíšená se právě navržená metoda – dále označená jako „kombinace MR + LL“ – zdá časově úspornější.

V případě, že je prioritou, aby se výstupní posloupnost skládala ze stoprocentně ověřených prvočísel, je možné použít metodu „kombinace MR + LL“ znázorněnou na obrázku 4.1, kde je zakomponována přímo ve schématu generátoru. Je ale třeba přihlédnout k faktu, že takovýto postup by generoval pouze Mersennova prvočísla a těch není příliš mnoho. Tato skutečnost by mohla znemožnit zaručení nepředvídatelnosti, která je základním požadavkem na generátor náhodné posloupnosti.

4.2 Postup zvolený k implementaci

Porovnání v předchozí sekci nastiňuje stupnici jednotlivých algoritmů od nejvhodnějších po nejméně vhodné pro sestrojení generátoru.

Tabulka 4.3 ukazuje jednotlivé kombinace způsobu generování a testování prvočíselnosti a slovně hodnotí, zda je efektivní je zkoušet implementovat.

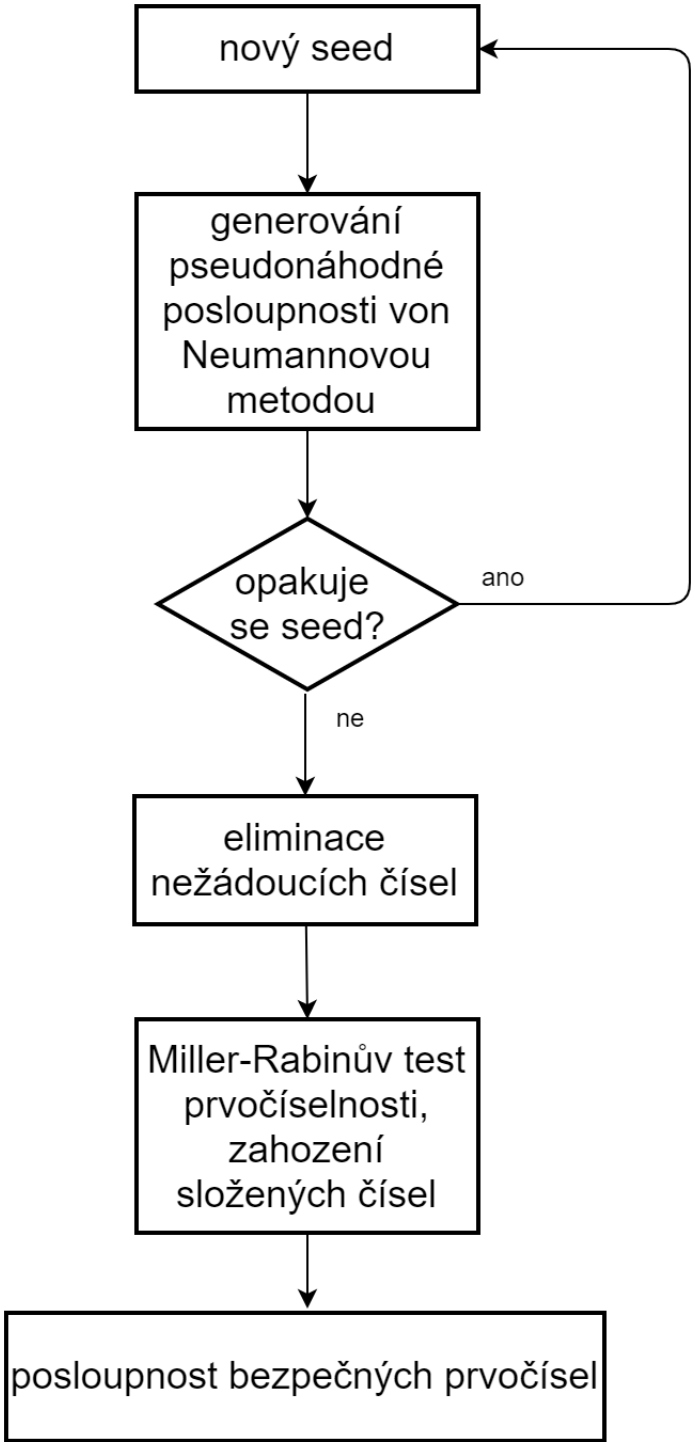
PRNG algoritmus	Miller-Rabinův test	Lucas-Lehmerův test	kombinace MR + LL
prostřední řády	nejrychlejší, vybraný k implementaci generátoru	pouze Mersennova prvočísla, nevhodný k implementaci	vhodný jedině pro generování Mersennových prvočísel
aditivní kongruence	pravděpodobně pomalejší, méně vhodný k implementaci	nevhodný k implementaci	nevhodný k implementaci
lineární kongruence	nevhodný k implementaci	nevhodný k implementaci	nevhodný k implementaci
Blum-Blum-Shrub	nevhodný k implementaci	nevhodný k implementaci	nevhodný k implementaci

Tab. 4.3: Matice navrhovaných řešení generátoru.

Jako optimální se jeví kombinace von Neumanovy metody generování (metody středních řádů) a Miller-Rabinova testu prvočíselnosti. Tento postup je graficky znázorněn na obrázku 4.2.

První seed bude na začátku pevně daný, bude zapsaný ve zdrojovém kódu, při detekci opakování se na jeho místo uloží nový, který bude získán z aktuálního x prostřednictvím multiplikativní kongruence. Multiplikativní kongruence, jinými slovy vynásobení konstantou, je použita z důvodu kompatibility s druhou mocninou, která je základem metody middle-square. Operaci $x \times x$ lze ve zdrojovém kódu snadno nahradit $x \times b$, má-li b stejnou bitovou délku jako x. V případě potřeby bude možné tuto část generátoru nahradit přebíráním seedu z externího TRNG.

Než začne test prvočíselnosti, bude na místě vyřadit z potenciálních výstupů nežádoucí čísla, jako jsou příliš krátká čísla a sudá čísla, na kterých je na první pohled poznat že nemohou být prvočísly.



Obr. 4.2: Schéma generátoru zvoleného k implementaci.

5 Implementace generátoru

Tato kapitola popisuje návrh implementace generátoru v programu Xilinx Vivado. Nejdříve rozebírá samotný zdrojový kód jazyka VHDL, do kterého je zatím slovně popsaný návrh přepsán. Podrobně se věnuje jednotlivým krokům Miller-Rabinova testu, který je časově nejnáročnější fází generování prvočísel. Dále jsou v této kapitole zdokumentovány testy jednotlivých částí generátoru provedené pomocí behaviorální simulace, kterou program Vivado umožňuje.

5.1 Provedení zvolených algoritmů v jazyce VHDL

Navržený generátor bude proměnné zpracovávat jako signály typu *std_logic_vector* a *unsigned*. Kromě proměnných obsahuje jednobitové signály *std_logic*, které přenášejí informace nezbytné k řízení procesů. Příkladem toho je signál *is_prime*. Ten je nastaven na hodnotu 1 v případě, že poslední zpracované x bylo vyhodnoceno jako pravděpodobné prvočíslo, a 0 v ostatních případech.

5.1.1 Metoda middle-square

K implementaci metody středních řádů je použita pomocná proměnná *x_power*. Má vyhrazen dvojnásobný počet bitů něž x a je stejného typu – *unsigned*. Proces generování dalšího čísla v posloupnosti má dva kroky.

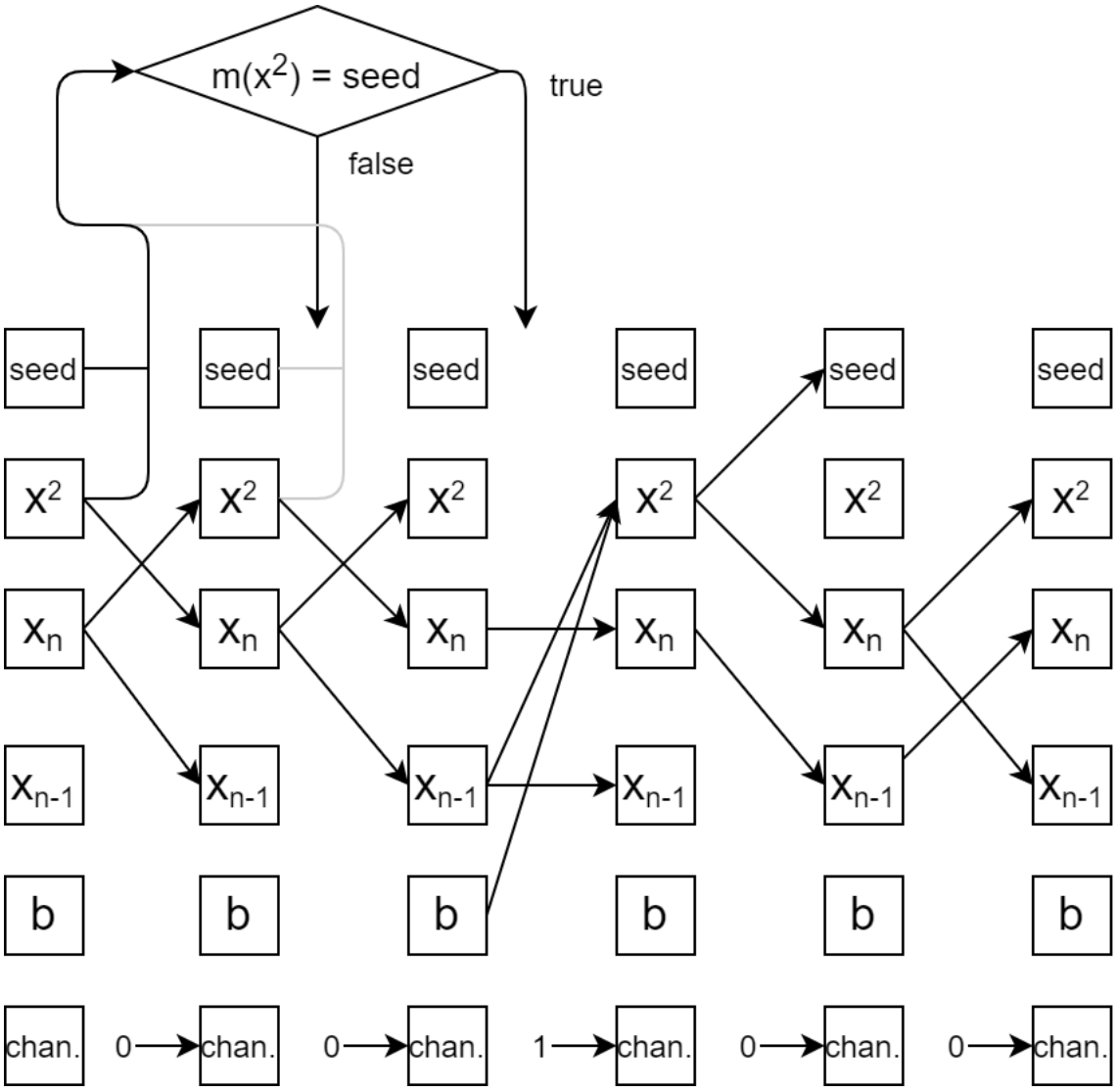
Nejdříve se položí $x\_power = x \times x$ a poté se signálu x přiřadí střední bity *x_power*. Přenos specifických bitů z jednoho signálu do druhého je v oblasti programovatelných obvodů jednoduchá operace, proto je k implementaci velmi vhodná.

5.1.2 Opakování seedu

Po vygenerování delší posloupnosti dojde k tomu, že se znovu objeví počáteční hodnota a celá řada se opakuje. Takové situaci zabrání podmínka if, která tuto situaci zachytí. Při zjištění tohoto nežádoucího stavu je třeba seed obnovit. To je možné prostřednictvím obměny algoritmu. Implementovaný generátor používá obměnu $x\_power = x \times b$, kde b je připravená konstanta stejného typu a délky jako x.

Po napravení tohoto nežádoucího stavu je ještě třeba uložit do signálu seed nové změněné x, aby oprava nebyla jen jednorázová. V kódu tento krok doprovází přepnutí pomocné proměnné *change* do stavu 1, aby nově nastavený seed mohl začít generovat řadu a nebyl ihned vyhodnocen jako chybový stav. Teprve po použití nového seedu se detekuje jeho opakování.

Schéma 5.1 ilustruje řadu procesů, které budou provedeny při detekci opakování seedu. Funkce $m(x^2)$ je výběr středních řádů proměnné *x_power* a podmínka $m(x^2) = \text{seed}$ je použita místo $x = \text{seed}$, aby k detekci došlo ještě před uložením nežádoucí hodnoty do signálu x. Podmínka je testována v každém cyklu, ve schématu je pro přehlednost zakreslena pouze jednou.



Obr. 5.1: Proces obměny algoritmu při opakování seedu.

Tato metoda není dokonalá. Může se stát, že se ve dvou posloupnostech založených na dvou různých seedech vyskytne stejné číslo. Tato událost se nedá efektivně zachytit, aniž by generátor ukládal všechna vygenerovaná čísla. To bohužel není možné, protože současná verze programu Vivado neumožňuje čtení a zápis do textových souborů. Je-li pro konkrétní aplikaci nezbytně nutné, aby se žádné číslo neopa-

kovalo, bude potřeba výstupní čísla ukládat a vzájemně porovnávat mimo generátor. Přesto metoda odstraňuje hlavní problém, tj. cyklické opakování celé posloupnosti.

5.1.3 Vynulování posloupnosti a okamžité opakování čísla

Dále může dojít k situaci, že prostřední cifry *x_power* by byly stejné jako předchozí x. K detekci tohoto stavu generátor použije proměnnou *x_prev*, do které se bude dočasně ukládat předešlé x_n a v rozhodovací větvi se porovná s x_{n+1}. V případě že se současné x rovná předchozímu, provede generátor změnu v podobě multiplikativní kongruence, podobně jako při opakování seedu.

Pomocí porovnání x s *x_prev* lze navíc detekovat vynulování posloupnosti. K vynulování dojde, když x_n je tak nízké, že jeho druhá mocnina má na místě všech středních cifer nuly. V situaci, kdy se x i *x_prev* budou rovnat nule, generátor provede opět změnu seedu, avšak tentokrát bude nový seed vycházet ze středních řádů čísla $c \times seed$, kde c je připravená konstanta.

5.1.4 Úprava délky čísel

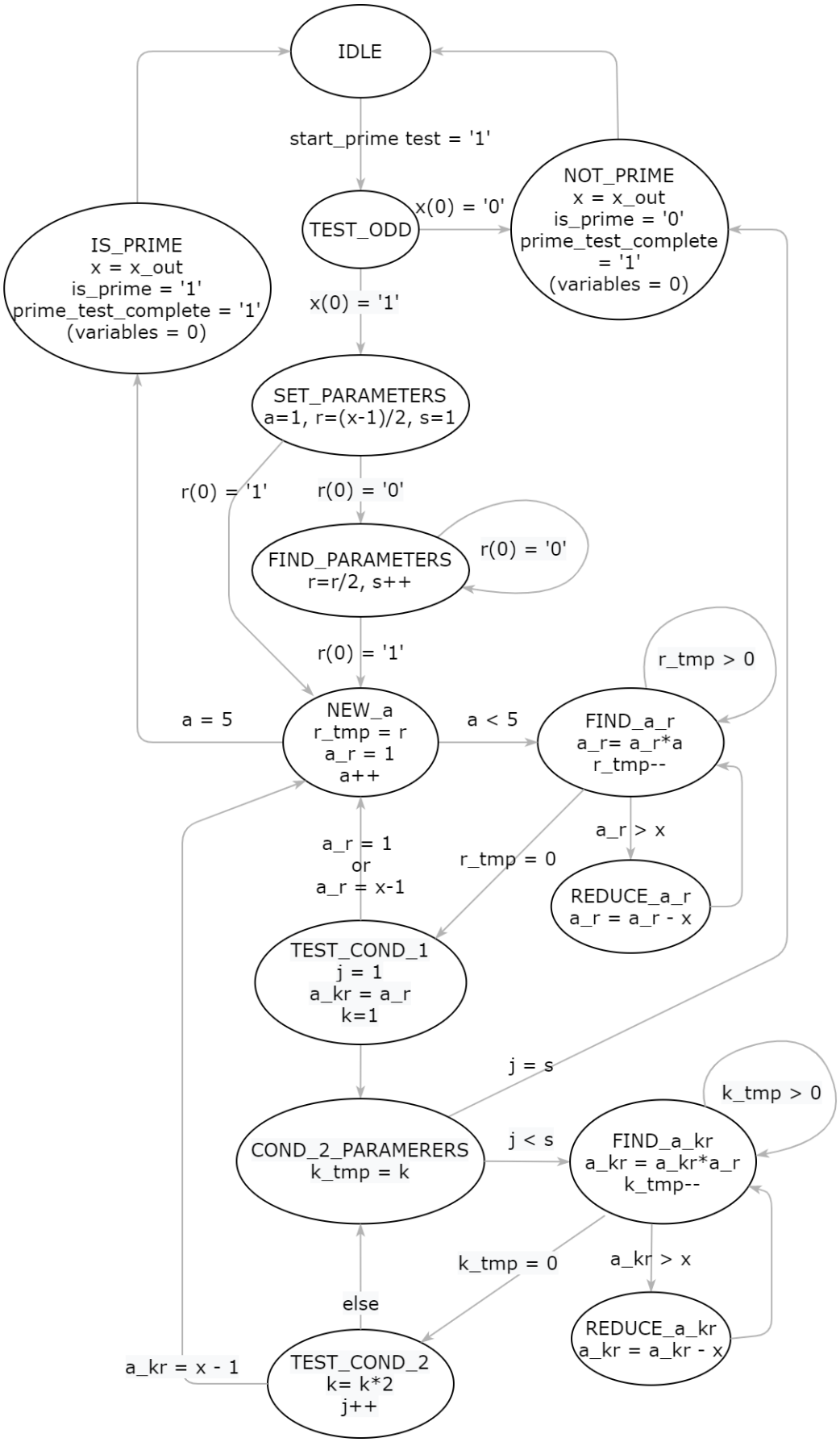
Jelikož bezpečná prvočísla vyžadovaná standarty pro jednotlivé kryptografické algoritmy mají minimální délky, je nežádoucí, aby měl generátor například na výstupu čísla kratší než 512 bitů v případě, že mají být tato prvočísla použita v algoritmu RSA s 2048bitovým n.

Z tohoto důvodu je před testem prvočíselnosti testována nerovnost s nejnižším číslem, které splňuje požadavky vybraného standartu. Jestliže je testované x menší, je vynecháno a nahrazeno následujícím v posloupnosti. Test bude předčasně ukončen, aby se nezdržoval vyhodnocováním čísla, které nesplňuje nastavené podmínky.

Další možností, která sice v konečné implementaci není, ale v praktickém užití by mohla být žádoucí, je zvýšení krátkého čísla x bitovým posunem, nebo přičtením konstanty. Bylo by potřeba podrobně analyzovat, jaký má taková operace dopad na dlouhodobou náhodnost posloupnosti.

5.1.5 Miller-Rabinův test v podobě stavového automatu

K otestování aktuálního čísla x Miller-Rabinovým testem prvočíselnosti generátor použije stavový automat. Především proto, že nezbytnou částí tohoto testu je mocnění a operace modulo, což jsou úkony, které se při návrhu obvodu musí rozložit na velké množství dílčích operací.



Obr. 5.2: Schéma stavového automatu pro realizaci Miller-Rabinova testu.

Na obrázku 5.2 je zakresleno schéma použitého automatu. Tato podkapitola slouží jako vysvětlivky ke grafické reprezentaci.

- IDLE: Výchozí stav, při kterém test začíná pro nové x, do tohoto stavu se automat vrátí po skončení testu.
- TEST_ODD: Ještě před začátkem samotného testu zajistí, aby sudá čísla – ta která končí nulou – byla označena za složená. Lichá čísla jsou předána dál k vlastnímu testování. Tento předtest jednoduše vyřadí přibližně polovinu nežádoucích výstupů.
- SET_PARAMETERS: Jak je uvedeno již v kapitole 2.5.2, Miller-Rabinův test potřebuje k testování parametry r a s, podle rovnice $n = 2^s r + 1$.
  V tomto případě je $n = x$. Parametry jsou nejdříve nastaveny na $r = x - 1$, $s = 0$ a následně je r děleno dvěma, dokud je výsledek celé číslo. s je zvětšeno o 1 pokaždé, když dojde k dělení r. Jelikož do tohoto stavu vstupují pouze lichá x, první $r = x - 1$ bude vždy sudé, a je tedy možné rovnou provést první dělení.
- FIND_PARAMETERS: Tento stav vykonává cyklické dělení zmíněné v předchozím bodě. Dělení vykoná v situaci, kdy r končí 0, tzn. dokud je sudé.
- NEW_a: Nyní jsou r a s nalezená a začíná test pro několik čísel a. Celý test je velmi časově náročný, hlavně protože operace mocnění a modulo vyžadují mnoho opakování procesů. Z každým dalším a se tento počet násobí (tím spíš, pokud jsou a velká čísla). Proto se tato implementace spokojí s otestováním tří a.
  První testované bude fakticky $a = 2$, protože pro $a = 1$ by podmínka $a^r \equiv 1 \pmod{x}$ platila vždy nehledě na hodnoty r a n.
  Jsou zde také nastaveny hodnoty $r\_tmp = r$ a $a\_r = 1$, se kterými pracuje další stav.
- FIND_a_r: Tento stav v kombinaci s následujícím slouží k výpočtu výrazu $a^r \mod x$. V každém průchodu tímto stavem se a_r nahradí $a\_r \times a$ a r_tmp se sníží o 1. Cílem tohoto stavu je r–krát vynásobit počáteční hodnotu 1 číslem a, jinými slovy provést operaci a^r.
- REDUCE_a: Tento stav je navštíven v případě, že a_r je vyšší než x a odečte x. Tím dojde k postupné modulaci a^r. Zároveň počet bitů potřebných pro signál a_r zůstane poměrně nízký – je zredukován vždy, když převýší délku x.
- TEST_COND_1: Po výpočtu $a\_r = a^r \mod x$ je možné vyhodnotit, zda platí první podmínka $a^r \equiv 1 \pmod{x}$. Jestliže platí, x je možným prvočíslem a test se opakuje pro jiné a.
  Zároveň je vhodné otestovat druhou podmínku $a^{2^j r} \mod x = x - 1$ pro $j = 0$ ($a^{2^0 r} = a^r$). V případě, že $a^r \mod x = x - 1$ je druhá podmínka splněna (stačí,

když platí pro jedno $0 \leq j \leq s-1$).

- COND_2_PARAMETERS: Sem automat přejde, když neplatí první podmínka, ani druhá podmínka pro $j = 0$. Tento stav připraví parametry pro počítání další operace modulo, tentokrát $a^{2^j r} \mod x$.
- FIND_a_kr: Tento stav slouží k výpočtu výrazu $a^{kr} \mod x$, přičemž $k = 2^j$ pro aktuální j. Z doposud prošlých stavů je již známo a^r, proto lze počítat $a^{kr} = (a^r)^k$

  Postup je podobný jako u stavu FIND_a_r, akorát zde se při průchodu stavem signál a_kr nahradí $a\_kr \times a\_r$. Tímto stavem má a_kr projít k–krát, avšak počet iterací pro dané j se bere z předchozí hodnoty k, tak aby operace $a^{kr} \mod x$ mohla použít výsledek z modula u předchozího j.

  Na první pohled se zdá jednodušší a_kr pro každé nové j vynulovat a pak mocnit aktuálním k, ale takový postup počítá několikrát to samé a dochází ke zbytečnému zdržení.

  Například v první iteraci je $j = 1$ a $k = 1$. Výchozí hodnota $a^{kr} = a^r$ je převzatá z testování první podmínky. $k\_tmp = k$ bude pro toto j rovno 1, takže vynásobení výchozí a^{kr} číslem $= a^r$ proběhne jen jednou. Výsledkem bude $a^r \times a^r = (a^r)^2$.

  U druhé iterace ($j = 2$) bude $k\_tmp = k = 2$ a dosavadní a^{kr} momentálně rovno $(a^r)^2$ bude číslem $= a^r$ vynásobeno dvakrát, s výsledkem $(a^r)^4$. To odpovídá $2^j = 2^2 = 4$.
- TEST_COND_2: Testování druhé podmínky: $a^{2^j r} \mod x = x-1$. V případě, že pro aktuální j ekvivalence platí, pak je testované a svědkem prvočíselnosti. Automat přejde do stavu NEW_a a opakuje test pro další a.
- IS_PRIME: Když projdou testem všechna zkoušená a a test stále nebyl předčasně ukončen, přejde automat do tohoto stavu, který přiřadí signálu is_prime hodnotu 1 a signálu x_out právě otestované x.
- NOT_PRIME: Stav pro předčasné ukončení testu. Jakmile neplatí pro liché číslo x ani první podmínka, ani druhá podmínka pro žádné j z povoleného intervalu, signál is_prime je nastaven na 0 a x_out přebere aktuální x, aby byla v simulaci vidět i čísla, pro která byl test neúspěšný.

5.2 Testování funkčnosti částí

Z podstaty zadání vyplývá, že generovaná čísla budou velmi vysoká, a nebude na první pohled zřejmá jejich prvočíselnost. Jinými slovy nepůjde rychle a jednoduše rozhodnout, zda má generátor skutečně na výstupu čísla požadovaných vlastností.

Výhodou použití typu *std_logic_vector*, respektive *unsigned* je jednoduše nastavitelná délka signálů. Toho se dá využít – nejdříve byla simulována verze generá-

toru s krátkými čísly. Při této simulaci lze snadno vyhodnotit, zda generátor a jeho části plní svůj účel.

5.2.1 Simulace metody middle square

V tabulce 5.1 je vypsáno prvních 5 výstupních čísel x generátoru využívajícího algoritmus výběru středních řádů pro 16bitová čísla. Seed využívaný pro tuto simulaci je 14 006.

x (hexadecimálně)	x	x^2
seed = 36b6	0011 0110 1011 0110 14 006	0000 1011 1011 0001 0100 1001 0110 0100 196 168 036
b149	1011 0001 0100 1001 45 385	0111 1010 1100 0110 0000 0110 1101 0001 2 059 798 225
c606	1100 0110 0000 0110 50 694	1001 1001 0010 1101 0100 1000 0010 0100 2 569 881 636
2d48	0010 1101 0100 1000 11 592	0000 1000 0000 0010 0110 0100 0100 0000 134 374 464
0264	0000 0010 0110 0100 612	0000 0000 0000 0101 1011 0111 0001 0000 374 544
05b7	0000 0101 1011 0111 1 463	0000 0000 0010 0000 1010 1000 1101 0001 2 140 369
20a8	0010 0000 1010 1000 8 360	0000 0100 0010 1010 0110 1110 0100 0000 69 889 600
2a6e	0010 1010 0110 1110 10 862	0000 0111 0000 1000 0100 0111 0100 0100 117 983 044
0847	0000 1000 0100 0111 2 119	0000 0000 0100 0100 1000 0011 1011 0001 4 490 161
4483	0100 0100 1000 0011 17 539	0001 0010 0101 0101 1101 1011 0000 1001 307 616 521

Tab. 5.1: Vygenerovaná čísla při seedu 14 006.

Tato verze generátoru zatím neobsahuje prostředky zabraňující opakování seedu a generování příliš krátkých čísel. Pro snadnější kontrolu jsou jednotlivá čísla převedena do desítkové soustavy. Díky tomu lze prověřit, že každé číslo je skutečně tvořeno středními ciframi druhé mocniny předchozího.

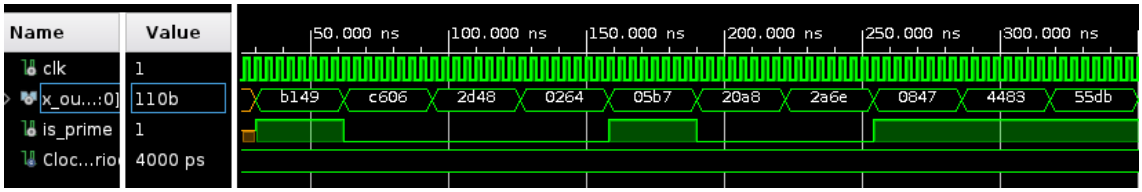
5.2.2 Simulace eliminace krátkých čísel

Pro tuto a následující simulaci je použita modifikovaná verze testu prvočíselnosti, který za prvočíslo označí každé liché číslo. To proto, aby se generátor nezdržel na časově náročné operaci modulo a vygeneroval rychle delší posloupnost čísel. Díky tomu bude možné sledovat případy, kdy vstupem stavového automatu bude nežádoucí číslo.

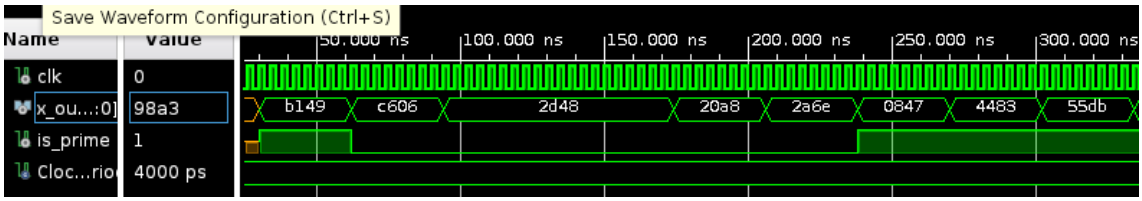
Na snímku z behaviorální simulace 5.3 jsou vidět generovaná čísla odpovídající tabulce 5.1. Některá z nich, jako třeba hned čtvrté v řadě, začínají nulou. Znamená to, že minimálně 4 jejich nejvýznamnější bity jsou obsazeny nulami. Taková čísla nelze považovat za 16bitová. Aby generátor dodržoval nějaké meze ohledně délky výstupních čísel, je potřeba eliminovat krátká čísla. Simulujme situaci, kdy mají výstupní čísla mít délku v rozmezí 13-16 bitů.

Aby byla čísla nižší než $0001000000000000_2 = 1000_{16} = 4096_{10}$ vyřazena ještě před testováním, je do stavového automatu přidán krok, který ještě před posuzováním, zda je číslo sudé či liché, otestuje nerovnost $x < 0001000000000000_2$.

Na snímku 5.4 již čísla začínající nulou nejsou vidět, protože nejsou předána výstupnímu signálu. Číslo 05b7 již není testováno, před eliminací krátkých čísel bylo vyhodnoceno jako liché (řádek *is_prime*). Kdyby se s ním prováděl Miller-Rabinův test, došlo by k výraznému zbytečnému zdržení.



Obr. 5.3: Simulace bez eliminace krátkých čísel.



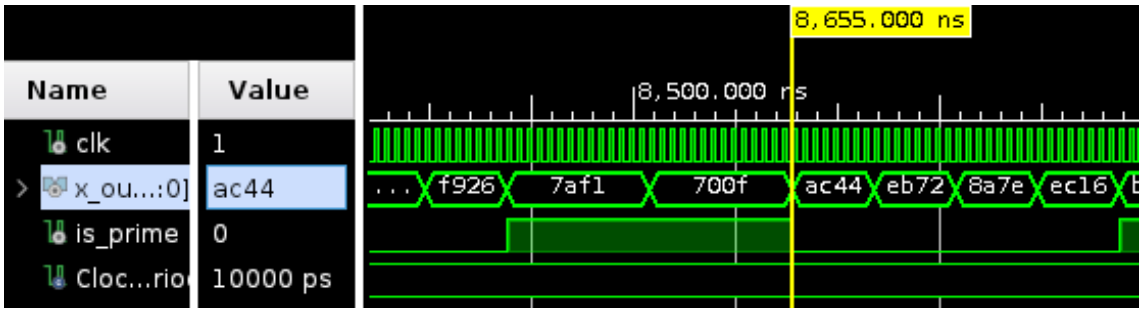
Obr. 5.4: Simulace s eliminací krátkých čísel.

5.2.3 Simulace opakování seedu

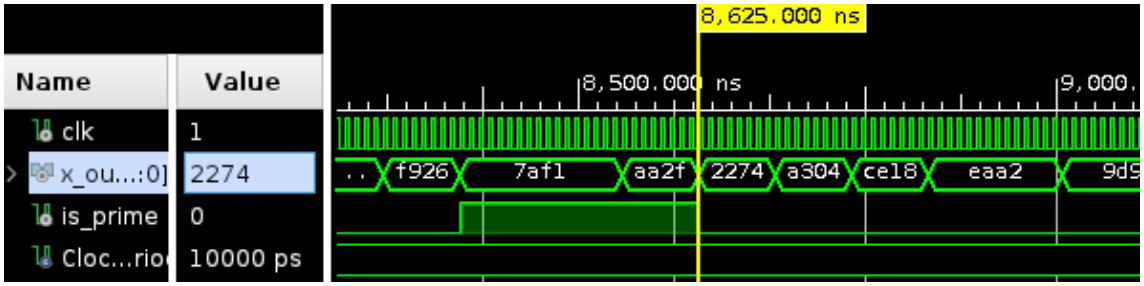
Na této verzi generátoru, který se nezdržuje na Miller-Rabinově testu lze zároveň vyzkoušet, jak bude generátor reagovat na opakování seedu.

Při zkoušce generátoru se seedem $ac44_{16} = 44100_{10}$ došlo k situaci, že po uplynutí času přibližně 3 450ns se posloupnost (ac44, eb72, 8a7e,... 7af1, 700f) opakuje, viz snímek 5.5. Tomu lze předejít úpravou prvního procesu. Když se před operaci $x_{i+1} = f(x_i)$ přidá rozhodovací větev if-else, která zachytí událost $x_i = seed$, při které se místo standartního umocnění číslo x_i vynásobí připravenou konstantou b. To ilustruje schéma 5.1.

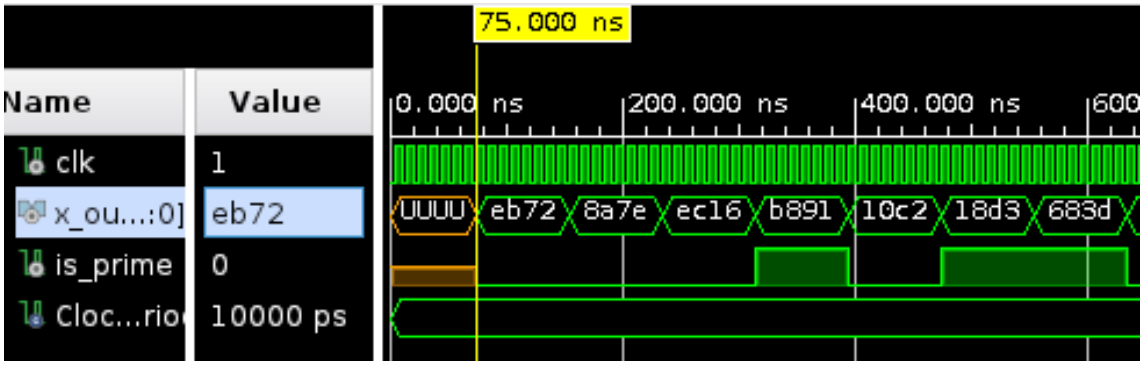
Na snímku z behaviorální simulace 5.6 je vůči předchozímu 5.5 vidět rozdíl v čísle, které následuje po 7af1.



Obr. 5.5: Simulace beze změny při opakování seedu.



Obr. 5.6: Simulace se změnou při opakování seedu.



Obr. 5.7: Referenční snímek toho, jak vypadají obě simulace na začátku.

5.2.4 Simulace s Miller-Rabinovým testem

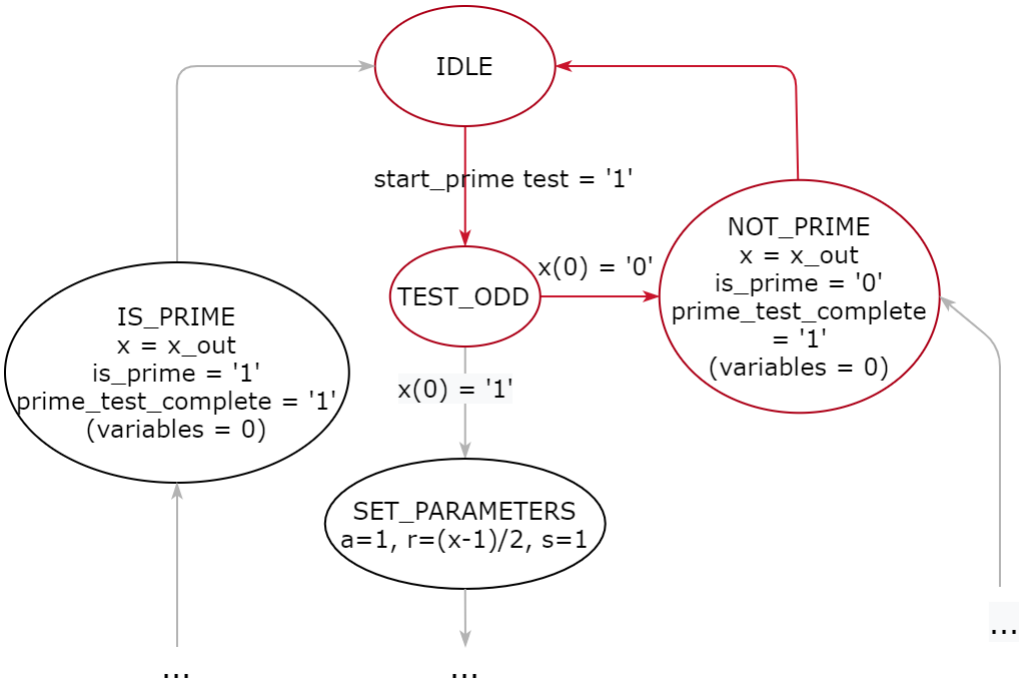
Po přidání stavového automatu (viz schéma 5.2) do zdrojového kódu generátoru je důležité otestovat, zda test skutečně pracuje zamýšleným způsobem. K tomu je použito krokování, které program Vivado nabízí. Umožňuje zobrazit aktuální stav jednotlivých objektů při behaviorální simulaci. Signály typu *std_logic_vector* a *unsigned* ukazuje v hexadecimální podobě, proto následující odstavce uvádí čísla hexadecimálně, s výjimkou čísel s, j a i, která jsou typu *integer*. Čísla a a v mnoha případech i k, jsou tak nízká, že jejich desítková podoba odpovídá šestnáctkové.

Kroky stavového automatu pro sudé číslo

Test bude mít nejkratší průběh pro sudá čísla. Vyhodnotí je hned na začátku jako složená, takže projdou jen stavy IDLE, TEST_ODD a NOT_PRIME.

Například hned na třetím řádku v tabulce 5.1 je vygenerované číslo $c606_{16} = 50694_{10} = 1100011000000110_2$. Když je vygenerováno a je spuštěn test prvočíselnosti, přejde automat do stavu TEST_ODD. Tento stav testuje, zda platí rovnost $x(0) = 0$, tedy jestli bit v poli x s indexem 0 je nastaven na hodnotu 0. V případě čísla c606 tato rovnost platí. V následujícím taktu automat přejde do stavu NOT_PRIME a test je označen za ukončený. Otestované číslo c606 je předáno na výstup generátoru společně s informací, že se jedná o složené číslo ($is\_prime = 0$).

Od konce testu předešlého čísla uplynulo pouze 8 taktů hodin.



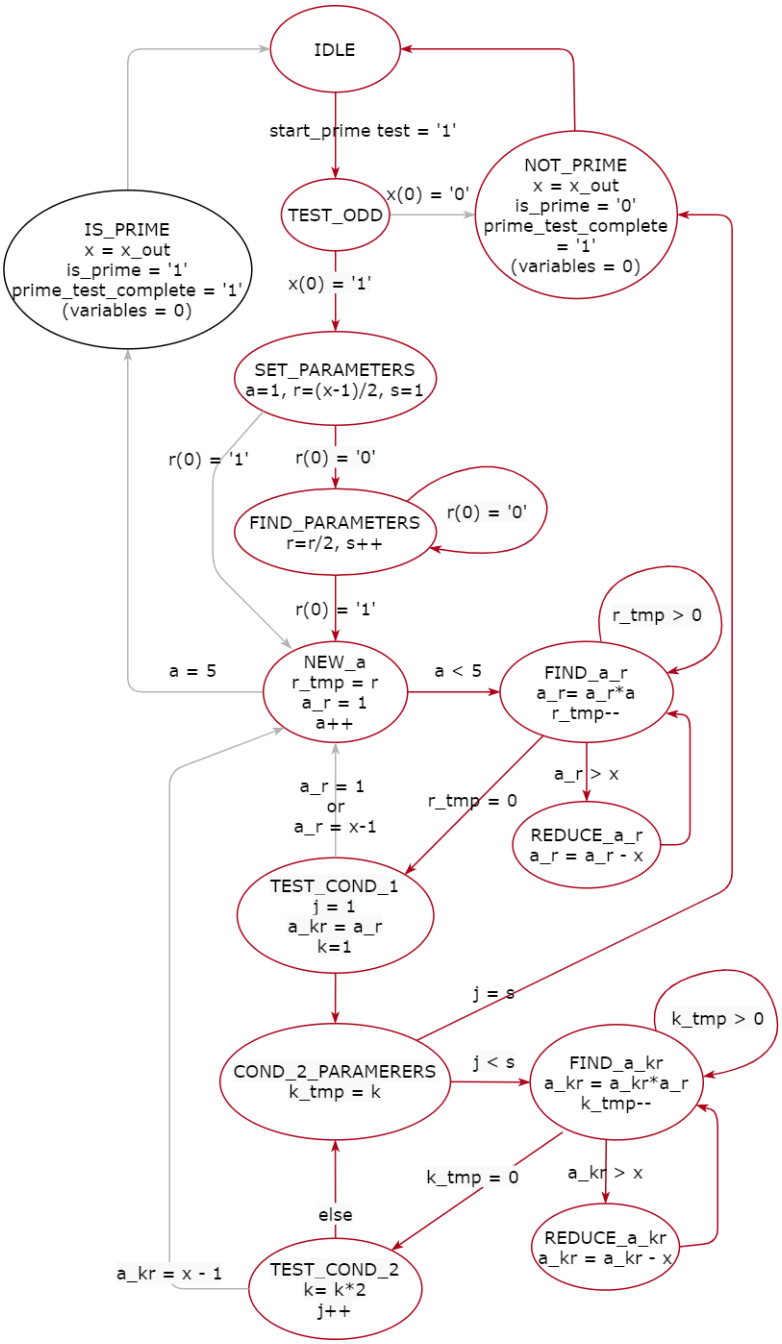
Obr. 5.8: Schéma navštívených stavů v případě, že x je sudé.

Kroky stavového automatu pro složené číslo

Hned první vygenerované číslo posloupnosti (v tabulce 5.1 na druhém řádku) je složeným a zároveň lichým číslem – z desítkové formy čísla je to poznat na první pohled, jelikož končí číslicí 5. Proto je číslo b149 vhodné k podrobnému otestování implementace Miller-Rabinova testu. Program Vivado na výpisu stavu signálů zobrazí x =b149. Po průchodu stavy IDLE, TEST_ODD a SET_PARAMETERS je $a = 2$, $s = 1$ a $r = 58a4$, což odpovídá $\frac{x-1}{2} = \frac{b149-1}{2}$, podle rovnice $x = r2^s + 1$.

Následuje stav FIND_PARAMERERS. Zde automat setrvá do nalezení parametrů $r = 1629$, $s = 3$. Přejde do stavu NEW_a, který přiřadí signálu a_r počáteční hodnotu 1. Následují stavy FIND_a_r a a REDUCE_a, které slouží k nalezení $a^r \mod x$. O mnoho taktů později – po r-násobném vynásobení čísla a (které je nyní rovno dvěma) je signál a_r roven číslu 0eb2. Protože se nerovná 1, je první podmínka vyhodnocena jako neplatná.

Zároveň se 0eb2 nerovná ani 1b48 ($= x - 1$), takže druhá podmínka neplatí pro $j = 0$. Zbývá otestovat druhou podmínku pro $j = 1$, případně $j = 2$. Automat nastaví $j = 1$ a přejde do stavu TEST_COND_2 a následuje další série násobení a modulace. Je zjištěno, že ani pro jedno z možných j neplatí nyní vyčíslená rovnost $a^{2^j r} \mod x = x-1$ a automat se připraví k testování $j = 3$. V té chvíli však přestane platit $j < s$ a celý test je ukončen přechodem do stavu NOT_PRIME.



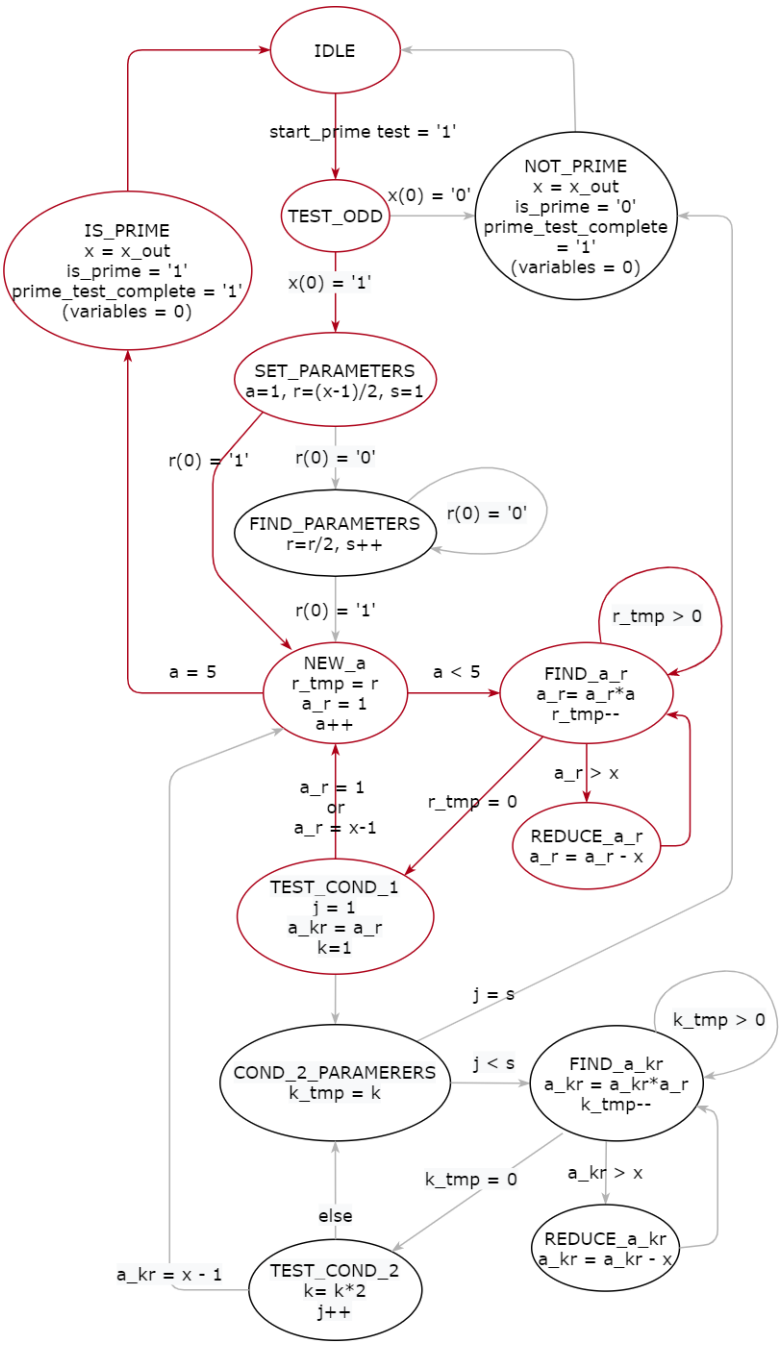
Obr. 5.9: Schéma navštívených stavů v případě, že x je liché složené číslo.

Kroky stavového automatu pro prvočíslo

První prvočíslo v řadě výstupů 16bitového generátoru je $4483_{16} = 17539_{10}$. Nalezené parametry pro Miller-Rabinův test jsou $s = 1$ a $r = 2241$ Pro první zkoušené $a = 2$ je vypočítáno $a\_r = a^r \mod x = 2^{2241} \mod 4483 = 4482$. To splňuje druhou podmínku pro $j = 0$: $x - 1 = 4483 - 1 = 4482$. To samé platí pro druhé testované a: $a\_r = 3^{2241} \mod 4483 = 4482$. Pro třetí a vyjde $a\_r = 4^{2241} \mod 4483 = 1$, což

splňuje pro změnu hned první podmínku.

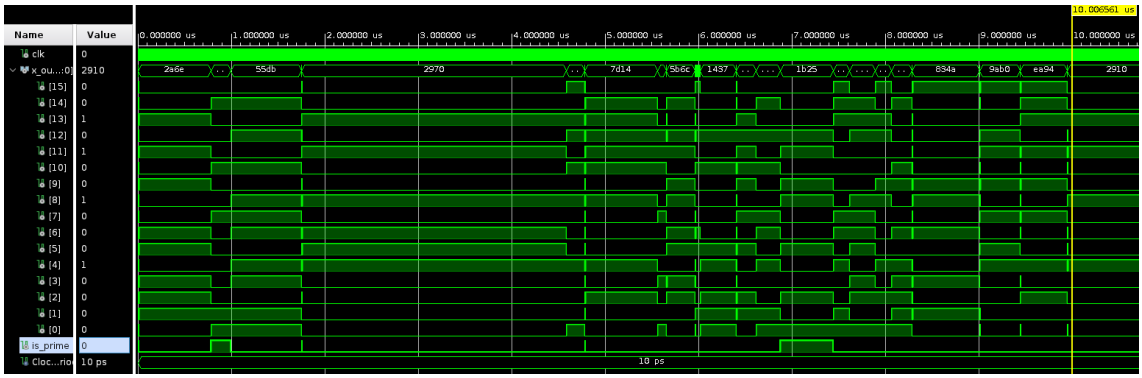
Na obrázku 5.10 jsou barevně vyznačeny stavy, kterými automat prošel při testování čísla 4483. V desítkové soustavě se jedná o číslo 17 539, jehož prvočíselnost byla ověřena webovou aplikací WolframAlpha.



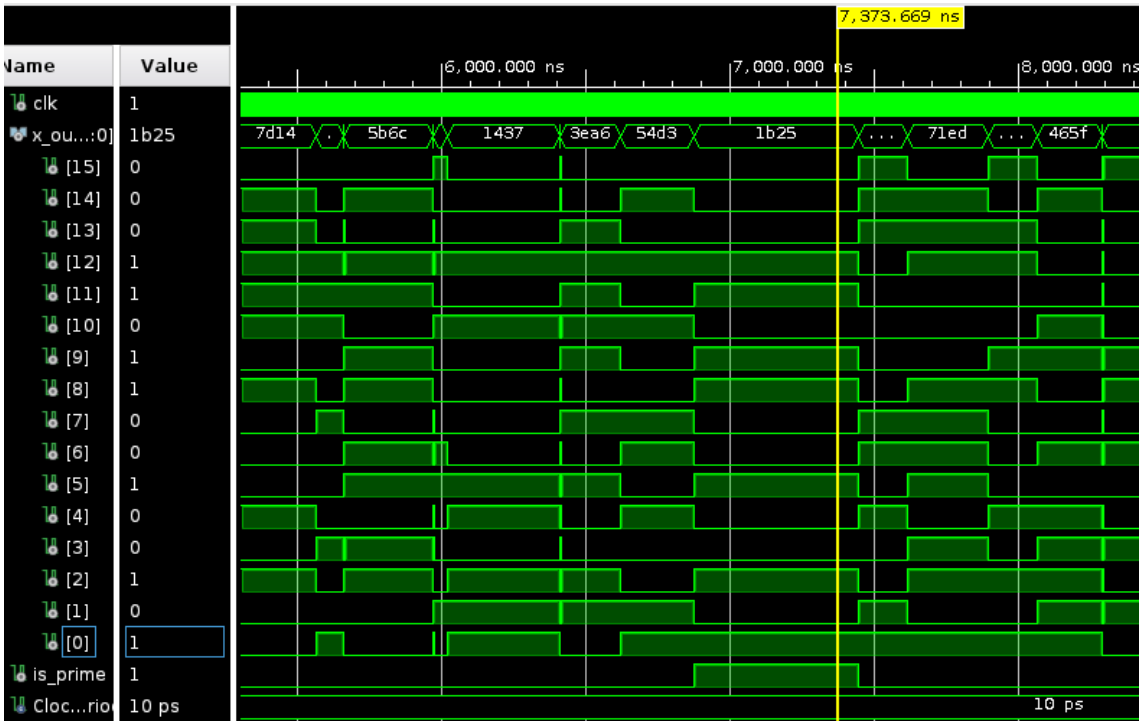
Obr. 5.10: Schéma navštívených stavů v případě, že x je prvočíslo 17 539.

5.3 Implementace generátoru pro 16bitová čísla

Tato podkapitola shrnuje chování generátoru, který obsahuje všechny doposud otestované části, zpracovává čísla dlouhá 16 bitů a začíná seedem 36b6. Takto naprogramovaný design byl úspěšně syntetizován.



Obr. 5.11: Simulace 16bitového generátoru, snímek 1.



Obr. 5.12: Simulace 16bitového generátoru, snímek 2.

Snímek 5.11 zachycuje behaviorální simulaci prvních 10μs při periodě hodin 10ps. Pozorování tohoto registru jako celku není příliš informativní. Každé číslo se různě

dlouho testuje, proto jsou některá čísla na výstupu velmi dlouho a jiná zase tak krátce, že se nevejdou do popisku. Snímek 5.12 ukazuje o něco detailnější pohled v okolí čtvrtého nalezeného prvočísla 1b25.

V tabulce 5.2 je vypsáno prvních 30 vygenerovaných čísel, prvočísla jsou zvýrazněna tučně. V prvních 8μs jsou vygenerována 4 prvočísla. Při simulaci trvající 35μs, tedy 3 500 000 taktů hodin, je celkem nalezeno 10 prvočísel. Konkrétně 4483, cb29, 110b, 1b25, b005, a9df, 8509, b713, df99 a 2803. Všechna jsou skutečně prvočísly, podle ověření na webu WolframAlpha.

x_{16}	x_{10}	x_{16}	x_{10}	x_{16}	x_{10}
b149	45 385	b511	46 353	a5f5	42 485
c606	50 694	**110b**	**4 363**	95bc	38 332
2d48	11 592	2276	8 822	9462	37 986
20a8	8 360	a38e	41 870	1437	5 175
2a6e	10 862	7e22	32 290	98a3	39 075
4483	**17 539**	257c	9 596	ddd8	56 792
55db	21 979	7d14	32 020	3ea6	16 038
cb29	**52 009**	1c89	7 305	54d3	21 715
3a0c	14 860	2e41	11 841	**1b25**	**6 949**
2970	10 608	5b6c	23 404	e0d3	57 555

Tab. 5.2: Simulace generátoru 16bitových čísel, seed = 36b6.

5.4 Implementace generátoru pro 256bitová čísla

Při mnohonásobném zvýšení délky zpracovávaných čísel lze kód pro generátor stále syntetizovat, ale funkčnost se zkouší těžko, protože operace s velkými čísly trvají mnohokrát déle. Samotné umocnění a_r může zabrat až $\frac{x-1}{2}$ průchodů stavem FIND_a_r. Pro nejvyšší možné x by to bylo více než $5,789 \times 10^{76}$ opakování procesu.

5.4.1 Návrh optimalizace

Při nejmenší nastavitelné periodě hodin – 10ps – po několika sekundách v simulaci a desítkách minut v reálném čase stále probíhá test prvního lichého čísla. Protože číslo a je možné volit náhodně, lze zajistit, aby bylo vždy mocninou dvou. V takovém případě bude možné operaci vynásobení číslem a vykonat jako bitový posun vlevo. Díky tomu bude možné v jednom průchodu stavem FIND_a_r provést vynásobení dvěma tolikrát, o kolik bitů je a_r delší než x. Po této úpravě bude nejdelší částí testu operace modulo, respektive průchod stavem REDUCE_a.

6 Zhodnocení vlastností navrhované implementace

Tato kapitola porovnává vlastnosti navržené a simulované implementace generátoru s očekáváním odvozeným z nastudovaných zdrojů. Komentuje, jak se v jednotlivých ohledech liší od ideálního generátoru. Požadavky na ideální generátor jsou: Nepredikovatelnost, rovnoměrné rozložení, nezávislost a vysoká rychlost. Generátor prvočísel klade navíc požadavek na délku čísel a samozřejmě prvočíselnost.

Uvedených třicet vzorků v tabulce 5.2 je příliš málo k provedení spolehlivé statistické analýzy. Navíc skutečně užitečnými výstupy jsou jen prvočísla určité délky. S tak specifickou množinou nelze prokázat že je rozložení skutečně rovnoměrné a je třeba se spolehnout, že běžně používané metody generování poskytují dostačující rovnoměrnost. Nezávislost v implementovaném generátoru zajišťuje výběr cifer. Ostatní vlastnosti jsou rozebrány podrobně v následujících podkapitolách.

6.1 Bezpečnost navržené implementace

Bezpečnost generátoru má dva aspekty. Aby byl prvočísla použitelná pro zabezpečování dat kryptografickými prostředky, mají být jednak dlouhá a jednak se nemají používat opakovaně.

6.1.1 Délka generovaných čísel

Implementace popsaná v kapitole 5.4 generuje čísla s délkou mezi 192 a 256 bity. Taková prvočísla naleznou v současnosti využití v algoritmech založených na eliptických křivkách a částečně i v DSA. Díky tomu že jsou čísla typu *std_logic_vector* se dají i zvýšit a generátor lze drobnou změnou v zdrojového kódu škálovat pro větší čísla. To vše je ale na úkor času.

6.1.2 Nepředvídatelnost

Dokud nedojde k nutnosti změny seedu, zaručuje von Neumannova metoda uspokojivou míru náhodnosti. Čím vyšší čísla se zpracovávají, tím menší je pravděpodobnost, že k opakování dojde tak brzy, že ovlivní průběh generování. Přesto generátor počítá se stavem opakování seedu a zacyklení je připraven oddálit pomocí multiplikativní kongruence s nachystanými konstantami.

V jedné ze simulací se vyskytla nečekaná událost: seed se podruhé nevyskytl a posloupnost se přesto po nějaké době začala opakovat. Tato situace je v rozporu

s předpokladem o algoritmu middle-square, že k zacyklení dojde v okamžiku opakování seedu. Pravděpodobně je to důsledek toho, že generátor pracuje s binárními čísly, a nikoliv s desítkovou soustavou, se kterou počítá citovaná literatura. Tento problém by nevznikal, kdyby generátor pracoval s integery. Protože cílem je generátor, který zpracovává čísla delší než 40 bitů, použití integerů je zcela nevhodné. Pro implementovaný generátor to znamená, že opatření proti opakování seedu v nejhorším případě funguje jen pro odstranění prvního zacyklení. Opakování posloupnosti tak oddálí ale nezabrání.

Při praktické aplikaci generátoru k dlouhodobému využití by bylo vhodné výstupy ukládat, například do utajeného souboru připojeného k výstupu generátoru. Čísla by se tak dala porovnávat a při detekci opakování by se mohl seed změnit.

6.1.3 Spolehlivost

V neposlední řadě musí generátor bezpečných prvočísel zaručit určitou míru jistoty, že jím generovaná čísla jsou skutečně prvočísly. Pro Miller-Rabinův test prvočíselnosti platí, že každé a, pro které platí podmínky testu, zaručuje prvočíselnost testovaného x s pravděpodobností 75 %. Šance chyby při otestování tří a je $\frac{1}{4^3} = 0{,}015625 \Rightarrow 1{,}5625$ % Vzhledem k porovnání s časovou náročností je úspěšnost s pravděpodobností přesahující 98 % uspokojivá.

Jestliže praktické uplatnění generátoru požaduje vyšší jistotu, lze jednoduše změnit počet testovaných a. Již při čtyřech a by byla spolehlivost přes 99,6 % .

6.2 Časová náročnost navržené implementace

6.2.1 Délka testu prvočíselnosti pro 16bitová čísla

Při simulaci s malými čísly lez pozorovat rozdíly v délce testování a závislost rychlosti testu na vlastnostech čísel. Dále je rozvedeno chování lichých čísel, protože pro sudá se Miller-Rabinův test zcela vynechává. Obecně pro všechna čísla platí, že čím delší jsou, tím déle trvá testování první podmínky. Doba testování vzroste, pokud dojde i na testování druhé podmínky pro $j > 0$. K tomu ale nemusí dojít u všech čísel – příkladem je prvočíslo 4483.

Prvočísla se zpravidla netestují tak dlouho jako složená čísla, protože stačí, aby se pro ně prokázala jedna z podmínek. K důkazu složenosti je naopak nutné projít všemi definovanými stavy.

Několik příkladů vygenerovaných složených čísel potvrzuje předpoklad, že čím vyšší prvočinitele má složené číslo a čím více jich je, tím obtížnější je činitele nalézt a dokázat tak složenost. Následující vygenerovaná čísla jsou převedena do desítkové

soustavy, rozložena na prvočísla a je uveden čas, po který byla testována. Čas jedné periody hodin byl nastaven na 10ps.

- $1437_{16} = 5175_{10} = 3 \times 3 \times 5 \times 5 \times 23$ – cca 50ns,
- $2e41_{16} = 11841_{10} = 3 \times 3947$ – cca 100ns,
- $1c89_{16} = 7305_{10} = 3 \times 5 \times 487$ – cca 750ns,
- $b511_{16} = 46353_{10} = 3 \times 15451$ – téměř 3000ns.

6.2.2 Délka testu pro vysoká čísla

Jak už předesílá kapitola 5.4, generátor upravený pro vysoká čísla nebyl detailně testován. Jednak proto, že u tak vysokých čísel nejde v přijatelném čase prokázat prvočíselnost, jednak kvůli zdlouhavé simulaci. Na základě simulací s menšími čísly lze předpokládat, časová náročnost bude slabším místem implementovaného generátoru. Testování prvočíselnosti se však bez dlouhých operací mocnění a modulo neobejde.

Závěr

Práce porovnává několik algoritmických metod generování přirozených čísel především z hlediska rychlosti. Jako nejrychlejší je zhodnocena von Neumanova metoda prostředních řádů. Ta je také navržena k implementaci. Pro hardwarový generátor je velmi vhodná, protože na platformě FPGA patří výběr specifických cifer k jednoduchým operacím, na rozdíl od násobení, mocnin, nebo operace modulo.

Práce také hodnotí výhody a nevýhody dvou různých testů prvočíselnosti – Miller-Rabinova a Lucas-Lehmerova testu. Pro implementaci generátoru je vybrán Miller-Rabinův test. Postup s použitím Lucas-Lehmerova testu by byl naprosto spolehlivý, ale generoval tak málo čísel, že se k implementaci generátoru spíše nehodí. Proto bude potřeba spokojit se s určitou nejistotou ohledně prvočíselnosti výstupu generátoru.

Následuje popis implementace ve VHDL. Von Neumannova metoda středních řádů je zkombinována s metodou multiplikativní kongruence, aby generátor dokázal napravit nežádoucí situace, jako je opakování seedu, okamžité opakování čísel v posloupnosti, nebo zbytečné testování příliš krátkých čísel, která nebudou odpovídat bezpečnostním požadavkům. Miller-Rabinův test je navržen v podobě stavového automatu, který je v kapitole 5.1.5 detailně popsán slovně i schematicky. Test je aplikován jen na lichá čísla, aby se ušetřil čas testování sudých, která se v binární podobě dají snadno detekovat.

Testování funkčnosti částí i celku generátoru proběhlo pomocí testbenche prostřednictvím behaviorálních simulací v programu Vivado. K testování byla použita verze, která generuje 16bitová čísla. Délku čísel lze jednoduše měnit přímo ve zdrojovém kódu. S čísly délky požadované pro kryptografické aplikace se generátor dost dobře testovat nedá. Z podstaty kryptograficky bezpečných čísel vyplývá, že se s nimi nedají složité matematické operace provádět v krátkých časových intervalech.

Na závěr práce hodnotí vlastnosti generátoru, který byl v programu Vivado simulován a syntetizován. Generátor bude mít na výstupu prvočísla s pravděpodobností vyšší než 98 %. S čísly délky 16 bitů se opakování seedu vyskytlo přibližně po stovce testovaných čísel. Se zvýšením počtu cifer by délka použitelné sekvence měla růst exponenciálně. V kombinaci s metodou multiplikativní kongruence poskytuje metoda středních řádů přijatelnou náhodnost. Míra nepředvídatelnosti by se však dala vylepšit připojením TRNG ke vstupu generátoru a úložiště čísel k výstupu. Z hlediska rychlosti generátoru jsou nejslabším místem generátoru operace mocnění a modulo, bez kterých se ale Miller-Rabinův test nedá provést.

Pro praktickou aplikaci generátoru práce doporučuje jej doplnit databází k ukládání výstupů a mechanismem pro detekci a nápravu opakování čísel, které po dlouhodobém použití může nastat.

Literatura

[1] BURDA, Karel. *Aplikovaná kryptografie.* Brno: VUTIUM 2013. ISBN 978-80-214-4612-0

[2] Prvočísla. *Matematika.cz.* [online]. Brno: Nová média, 2014 [cit. 2020-12-09]. Dostupné z URL: <https://matematika.cz/prvocisla>.

[3] HAJNÝ, Jan. *Přednáška předmětu BZKR - Základy kryptografie 2.* [online prezentace]. VUT v Brně, 2018 [cit. 2020-12-09]. Dostupné z URL: <https://moodle-archiv.ro.vutbr.cz/course/view.php?id=199221>.

[4] HAJNÝ, Jan. *Přednáška předmětu BZKR - Základy kryptografie 4.* [online prezentace]. VUT v Brně, 2018 [cit. 2020-12-09]. Dostupné z URL: <https://moodle-archiv.ro.vutbr.cz/course/view.php?id=199221>.

[5] HAJNÝ, Jan. *Přednáška předmětu BZKR - Základy kryptografie 7.* [online prezentace]. VUT v Brně, 2018 [cit. 2020-12-09]. Dostupné z URL: <https://moodle-archiv.ro.vutbr.cz/course/view.php?id=199221>.

[6] ZEMAN, Václav. *Přednáška předmětu BPC-AKR - Asymetrické šifry I.* [online prezentace]. VUT v Brně, 2019 [cit. 2020-12-09]. Dostupné z URL: <https://moodle.vutbr.cz/course/view.php?id=210586>.

[7] HAJNÝ, Jan. *Přednáška předmětu BPC-CPT - Cryptologic Protocol Theory 4.* [online prezentace]. VUT v Brně, 2021 [cit. 2021-05-05]. Dostupné z URL: <https://moodle.vutbr.cz/course/view.php?id=224125>.

[8] BARKER, Elaine. *Recommendation for Key Management.* NIST SP 800-57 PART 1 REV. 5. Dostupné z URL: <https://csrc.nist.gov/Projects/Key-Management/key-management-guidelines>.

[9] RICCI, Sara. *Přednáška předmětu BPC-CPT - 5. Eliptic Curves in Cryptography.* [online prezentace]. VUT v Brně, 2021 [cit. 2021-05-05]. Dostupné z URL: <https://moodle.vutbr.cz/course/view.php?id=224125>.

[10] OCHODKOVÁ, Eliška. *Matematické základy kryptografických algoritmů.* Vytvořeno v rámci realizace projektu Matematika pro inženýry 21. století (reg. c. CZ.1.07/2.2.00/07.0332), 2011.

[11] Information Technology Laboratory, National Institute of Standards and Technology. *Digital Signature Standard (DSS).* FIPS PUB 186-4 2013 Dostupné

z URL:
<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[12] HAJNÝ, Jan. *Přednáška předmětu BZKR - Základy kryptografie 6.* [online prezentace]. VUT v Brně, 2018 [cit. 2020-12-09]. Dostupné z URL:
<https://moodle-archiv.ro.vutbr.cz/course/view.php?id=199221>.

[13] BARKER, Elaine a Quynh DANG. *Recommendation for Key Management.* NIST SP 800-57 PART 3 REV. 1. Dostupné z URL:
<https://csrc.nist.gov/Projects/Key-Management/key-management-guidelines>.

[14] HAJNÝ, Jan. *Přednáška předmětu BZKR - Základy kryptografie 5.* [online prezentace]. VUT v Brně, 2018 [cit. 2020-12-09]. Dostupné z URL:
<https://moodle-archiv.ro.vutbr.cz/course/view.php?id=199221>.

[15] BURGET, Radim. *Přednáška předmětu BPC-TIN - Vyčíslitelnost a složitost.* [online prezentace]. VUT v Brně, 2020 [cit. 2020-12-09]. Dostupné z URL:
<https://moodle.vutbr.cz/course/view.php?id=210601>.

[16] ZEMAN, Václav. *Přednáška předmětu BPC-AKR - Náhodná čísla* [online prezentace]. VUT v Brně, 2019 [cit. 2020-12-09]. Dostupné z URL:
<https://moodle.vutbr.cz/course/view.php?id=210586>.

[17] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, Mike Boyle. *Recommendation for the Entropy Sources Used for Random Bit Generation.* NIST Special Publication (SP) 800-90B, 2018 Dostupné z URL:
<https://csrc.nist.gov/CSRC/media//Publications/sp/800-90b/draft/documents/sp800-90b_second_draft.pdf>.

[18] Introduction to Randomness and Random Numbers. *RANDOM.ORG* [online]. Dublin: Randomness and Integrity Services, 2020 [cit. 2020-12-09]. Dostupné z URL:
<https://www.random.org/randomness/>.

[19] KNUTH, Donald Ervin. *The art of computer programming.* 3rd ed. Upper Saddle River, NJ: Addison-Wesley, c1998. ISBN 978-0-201-89684-8.

[20] ZEMAN, Václav. *Přednáška předmětu BPC-AKR - Hašovací funkce* [online prezentace]. VUT v Brně, 2019 [cit. 2020-12-09]. Dostupné z URL:
<https://moodle.vutbr.cz/course/view.php?id=210586>.

[21] HAJNÝ, Jan. *Přednáška předmětu BPC-CPT - Cryptologic Protocol Theory 2.* [online prezentace]. VUT v Brně, 2021 [cit. 2021-05-05]. Dostupné z URL: <https://moodle.vutbr.cz/course/view.php?id=224125>.

[22] GOLLOVÁ, Alena. *Testy prvočíselnosti - 19. a 20. přednáška z kryptografie* [online prezentace]. ČVUT - Katedra matematiky, 2020 [cit. 2021-05-05]. Dostupné z URL: <https://math.feld.cvut.cz/gollova/mkr/mkr9.pdf>.

[23] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL.* Praha: BEN - technická literatura, 2006. ISBN80-7300-198-5

[24] KUBÍČEK, Michal. *Úvod do problematiky obvodů FPGA pro integrovanou výuku VUT a VŠB-TUO.* VUT v Brně, 2014 [cit. 2021-05-05]. Dostupné z URL: <https://moodle.vutbr.cz>.

[25] YANG, Bohan. *True Random Number Generators for FPGAs.* ARENBERG DOCTORAL SCHOOL, Faculty of Engineering Science, 2018.

[26] D. Indhumathi Devi, S. Chithra, M. Sethumadhavan. *Hardware Random Number Generator Using FPGA.* Journal of Cyber Security and Mobility, 2019. Dostupné z URL: <https://www.riverpublishers.com/journal_read_html_article.php?j=JCSM/8/4/1>.

Seznam symbolů, veličin a zkratek

GCD	Nejvyšší společný dělitel – Greatest Common Divisor
RSA	Šifrovací algoritmus pojmenovaný podle autorů – Rivest, Shamir, Adleman
DSA	Algoritmus digitálního podpisu – Digital Signature Algorithm
NIST	Národní institut standardů a technologie – National Institute of Standards and Technology
ECC	Kryptografie založená na eliptických křivkách – Eliptic Curve Cryptography
TRNG	Skutečně náhodný generátor posloupností – True Random Number Generator
PRNG	Pseudonáhodný generátor posloupností – Pseudorandom Number Generator
FPGA	Field Programable Gate Arryay – programovatelné hradlové pole
VHDL	VHSIC Hardware Description Language – jazyk pro popis hardware
VHSIC	Very High Speed Integrated Circuits – vysokorychlostní integrované obvody
SRAM	Static Random Access Memory – statická paměť
CLB	Configurable Logic Block – konfigurovatelný logický obvod
QRNG	Kvantový generátor náhodných posloupností – Quantum Random Number Generator

Obsah přiloženého adresáře

V odevzdaném adresáři jsou dva projekty spustitelné v programu Vivado 2020.2.

Nejdůležitějšími soubory jsou v každém z projektů zdrojové kódy generátorů a k nim přiřazené testbenche – generator_16.vhd, generator_16_testbench.vhd, generator_256.vhd a generator_256_testbench.vhd. Tyto soubory lze samostatně importovat do nového projektu pro spuštění behaviorální simulace.

```
/ ..... kořenový adresář přílohy
├── generator_16 ..... projekt pro 16bitový generátor
│   ├── generator_16.cache
│   ├── generator_16.hw
│   ├── generator_16.ioplanning
│   ├── generator_16.ip_user_files
│   ├── generator_16.runs
│   ├── generator_16.sim
│   ├── generator_16.srcs
│   │   ├── constrs_1
│   │   └── sources_1
│   │       ├── generator_256.vhd
│   │       └── generator_256_testbench.vhd
│   └── generator_256.xpr
└── generator_256 ..... projekt pro 256bitový generátor
    ├── generator_256.cache
    ├── generator_256.hw
    ├── generator_256.ioplanning
    ├── generator_256.ip_user_files
    ├── generator_256.runs
    ├── generator_256.sim
    ├── generator_256.srcs
    │   ├── constrs_1
    │   └── sources_1
    │       ├── generator_256.vhd
    │       └── generator_256_testbench.vhd
    └── generator_256.vhd.xpr
```